BETER PROGRAMMEREN MET
BETA BASIC
en de
ZX SPECTRUM (+)

S.GIRARD en L.VERBOVEN

BETER PROGRAMMEREN MET
BETA BASIC
en de
ZX SPECTRUM (+)

S.GIRARD en L.VERBOVEN

TERMINAL SOFTWARE PUBLICATIES

Een uitgave van. Terminal Software Publicaties
Postbus 111, 5110 AC Baarle Nassau

1e druk oktober 1985

ISBN 90 6883 014 7

CIP GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Girard, Serge

Beter programmeren met Beta Basic en ZX Spectrum (+) /
Serge Girard en Luc Verboven. - Baarle Nassau : Terminal
Software Publicaties. - Tab.
Met reg.
ISBN 90-6883-014-7
SISO 365.3 UDC 681.3.06+800.92 Beta Basic UGI 650
Trefw.: Beta-Basic (programmeertaal) / programmeren
(computer) / ZX Spectrum (computer).

I N H O U D

VOORWOORD

Was u al een gelukkige bezitter van een Spectrum computer dan wordt het met Beta-Basic helemaal te gek.
Wij hebben Beta-Basic nu circa 1 jaar en vanaf dat moment hebben wij, ondanks dat de Spectrum-Basic niet slecht is, vastgesteld dat Beta-Basic nieuwe horizonten opent voor hen die met Spectrum Basic goed overweg kunnen. Sedert enige tijd hebben wij ontdekt dat er vraag is naar meer informatie . Meer dan dewelke die te vinden is in de Engelstalige handleiding. Hopelijk is dit boek een stap in de goede richting.

Alhoewel Beta-Basic circa 10K geheugen in beslag neemt zijn wij van mening dat dit gecompenseerd wordt door de vele nieuwe keywords en functies, waarvan u er vele in andere Basics zult aantreffen. Allemaal tegelijk zult u ze echter in zeer weinig Basics aan treffen.

De combinatie Spectrum (+) en Beta-Basic is dan ook ooit " de meest uitgebreide en krachtigste Basic ter wereld " genoemd en " zelfs de beste Basic mist een flink aantal commando's die Beta-Basic wel kent " (bron : RAM nr. 48 juli/aug, 1984)

U zult zich dan ook nieuwe programmeertechnieken aanleren die de moeite weerd zijn.

De onderwerpen in dit boek hebben wij gerangschikt in drie grote delen, te weten:

- uitleg en informatie over de keywords

- uitleg en informatie over de functies

- voorbeeldprogramma's

Bij de uitleg worden in veel gevallen voorbeelden gegeven, terwijl de voorbeeldprogramma's meer gebruiksklare routines en zelfs complete (kant en klaar) programma's bevat.

Bovendien vonden wij het nuttig om het geheugen van de Spectrum met Beta-Basic in scheme te brengen, evenals twee ley-outs van de toetsenborden met daarop aangeduid de nieuwe keywords en functies.

De programma's zijn eenvoudig in eenvoudig Beta-Basic geschreven, terwijl het gecombineerd gebruik ervan in verdere programma's steeds toeneemt.

Wij danken onze uitgever voor de medewerking en het vertrouwen dat hij in ons stelde en de mensen rondom ons voor het geduld dat zij met ons hebben gehad.

augustus 1985,
de auteurs.

2. ALGEMEEN BETA BASIC.

Het opstarten van Beta-Basic.

Beta-Basic heeft als doel de Spectrum-Basic uit te breiden om eenvoudiger en meer gestruktureerd te kunnen programmeren. De mensen van Betasoft zijn daar zeer goed in geslaagd. Beta-Basic 1.8. levert circa 30 nieuwe keywords, 8 verbeterde Spectrum-keywords, 8 nuttige variebelen, liefst 22 gedefinieerde funkties (TABEL) en dit is nog niet alles.

De cursorcontrole is aangevuld en GOTO, GOSUB en RETURN werken sneller den voorheen als de GOTO- en GOSUB-lijnen aan stuk terugspringen in het programma. RETURN naar lijn 9999 is nu niet zo snel als een RETURN naar lijn 1, wat in Spectrum-Basic niet het geval is. De combinatie Spectrum- en Beta-Basic brengt een zeer krachtige Basic tot stand!

LOAD Beta-Basic.

Op de Beta-Basic cassette kan een 16K- en een 48K-versie staan (1.0 uitgave) : voor het inlezen : LOAD "bb16" of "bb48", Maar heeft u de gepaste versie klaar om in te lezen, dan moet LOAD "" trouwens ook kunnen. LOAD "" moet ook voor 1.8 en 1.9 volstaan.

Eerst wordt een Basic-gedeelte ingelezen met de regels 0, 1 en 2 en daarna automatisch ook machinecode. Is het LOADen goed verlopen, dan krijgt u de copyright-boodschap bv. "Beta Basic 1.9. c Betasoft 1984 " en na enteren een 0-regel te zien, waarin de funkties van Beta-Basic zijn opgenomen, Regels 1 en 2 werden uitgewist nadat de copyright-boodschap verscheen. Met een MERGE"" kunt u ze toch inlezen. Dat is nuttig om er bv. nog wat meer bij te schrijven, maar in eerste instantie noodzakelijk voor het maken van een copie voor eigen gebruik, terwijl de oorspronkelijk cassette als backup-copy wordt weggeborgen.

DE BASICREGELS VAN BETA-BASIC.

```
1  LET rt = DPEEK (23730):
   RANDOMIZE USR 59904:
   SAVE "Beta Basic" LINE 2:
   POKE  DPEEK (23631) + 2, 181:
   SAVE "BB"CODE rt+1,65367-rt:
   STOP

2  CLEAR rt : LOAD "BB" CODE:
   CLS : RANDOMIZE USR 58419 :
   DELETE 1 TO 2
```

De verklaring van deze regels luidt als volgt.

De variabele rt of ramtop krijgt de inhoud van de systeem-variabele RAMTOP of DPEEK (23730) toegewezen, waarna RANDOMIZE USR machinecode aanroept op adres 59904. Deze blijkt de Beta-Basic 'af' te zetten.

SAVE "Beta-Basic" LINE 2 doet na het LOADen BB starten op LINE 2 en de POKE DPEEK (23631) + 2, 181 is er om automatisch een volgend programmedeel te SAVEn, zonder dat de Spectrum mededeelt 'Start tape, then press any key' .

Na het opladen van de basiclijnen van Beta-Basic, wordt onmiddellijk de tweede regel uitgevoerd, die het Basicgeheugen afbakent met CLEAR rt of: maak van rt de laatste byte die beschikbaar is voor Basic , zodat boven rt (vanaf rt+1) de machinecode kan ingelezen : LOAD "BB" CODE .

CLS : RANDOMIZE USR 58418 roept de copyright-boodschap op en start de Beta-Basic: u kunt nu tevens zien welke versie u bezit. Regels 1 en 2 worden gewist met DELETE 1 TO 2.

Er is nu ook een oplichtende cursor verschenen die in 1 stap naar het eind van de basicregel in de editmode kan verspringen indien cursor 7 wordt ingedrukt. De cursortoetsen 6 en 7 kunnen ook per regel verspringen. Zie hiervoor bij appendix 1 : ' de cursor controle codes'.

De systeemvariabele PIP op adres 23609 veroorzaakt de keyboardklik na het drukken van een toets. Omdat een lange PIP het indrukken van de toetsen vertraagt:

```
POKE 23609, 0      om de klik te doen verdwijnen, maar ook
POKE 23609, -2     om een nedrukkelijke klik te krijgen
POKE 23609, 3      voor een korte klik
POKE 23609, 10     voor een lange klik
```

Experimenteer met andere weerden en zoek Uw voorkeur uit.

Bij het inbrengan van een lijn verdwijnt de 0-lijn, zodat ze de listing van je programme's niet stoort. Enter nu NEW en u ziet de 0-lijn opnieuw verschijnen. Beta-Basic blijft dus beweerd bij NEW, doch het basic- en variabelengeheugen worden wel gewist of geCLEARd.

Het inlezen van programma's die niet in BB werden geschreven gebeurt met MERGE "naam" , went een LOAD-opdracht zou de 0-lijn wissen, zodat u de nieuwe funkties van BB niet kunt gebruiken. U krijgt dan als foutboodschap 2S, 'P FN without DEF', indien U toch een Beta-Basic funktie tracht op te roepen.

indien gewenst (om geheugen te besperen of om eigen functies te definieren) kunt U regei 0 er zelf ook uit halen met DELETE 0 TO 0. De Beta-Basic functies zijn dan niet meer te gebruiken.

Programma's met BB geschreven zullen de 0-lijn bevatten en kunnen dus wel geLOAD worden. Gewone Basic-programma's verlopen normaal wenneer zij in Beta-Basic worden opgenomen.

Worden GRAPHICS of UDG's (user-defined-graphics) gebruikt, dan dient de opdracht KEYWORDS 0 ingebracht. Deze roept de GRAPHICS weer terug. De keywords van BB 'overschrijven' immers de GRAPHICS, zolang KEYWORDS op 1 staat. We komen hier nog op terug.

Bij het eangaven van de formaten van de nieuwe keywords en funkties wijzen de tekens < en > erop det hun inhoud optioneel is en dus waggelaten kan worden. Ze behoren niet tot de werkelijke gebruiksvorm en dienen dus niet ingetikt te worden.

Als voorbeeld het algemene Formaat:

AUTO <lijnnummer> <,stapwaarde>

De opdrachten

AUTO
AUTO lijnnummer
AUTO lijnnummer, stapwaarde

zijn allen mogelijk. De komma in het tweede subformaat geeft aan dat de tweede optie <,stepweerde> niet kan gebruikt worden zonder het eerste deelformaat. AUTO ,stapwaarde is dus een foutief formaat. We treffen dit slechts aan bij AUTO en bij het uitgebreide formaat van ROLL en SCROLL vanaf versie 1.8.

Wordt een deelformeet aangegeven met de haakjes (an), dan behoren deze wel tot de eigenlijke gebruiksvorm en dienen ingetoets indien men in de opdrecht dit subformaat opneemt.

De toetsen welke dienen ingedrukt om de Beta-Basic keywords en Funkties te bekomen, worden telkens in de hoofding van elk hoofdstukje opgenomen. Bij voorbeeld :

GRAPHICS, (SYMBOL SHIFT + 1) om het keyword KEYIN op te roepen.
In deze 'toetsformaten' wijzen de komme's erop dat de toetsen na elkaar dienen ingedrukt. Het + teken geeft aan dat de toetsen tussen haakjes tesemen moeten worden ingedrukt.

In sommige van onze listings kunnen spaties voorkomen die niet noodzakelijk zijn (tenzij deze in stringvorm!), meer die wel de listing aanzienlijk kunnen verduidelijken. Het voorkomen van meerdere VAL "n" spaart per getal 3 bytes aan Basicgeheugen, terwijl de listing toch langer wordt, waardoor het programma vertraagt. Vele VAL "n" in o.m. een uitgebreide PRINT-instruktie of berekening kunnen het runnen ervan merkbaar vertragen.

SAVE Beta-Basic.

Het maken van copieen voor eigen gebruik van Beta-Basic (programma en de code) is vrij simpel. Indien wa van nul zouden beginnen, den zet U de Spectrum aan en geeft da directe opdracht:

MERGE "" en zet U de cessette-recorder aan. Vervolgens zal alleen het programma-gedeelte ingeladen worden. Edit regel 2 en verander het laatste gedeelte (DELETE 1 TO 2) in STOP.

Geef nu GOTO 2 en het machinecode gedeelte zal ingeladen worden. Na het inladen baschikt U over de mogelijkheid om een copie te maken door GOTO 1 te geven. U kunt ook beginnen met programmeran en na afloop het geheel (uw programma en de mechinecode) SAVEn door eveneens GOTO 1 te geven.

Na het SAVEn kunt u desgewenst NEW geven en opnieuw starten met programmeren. Het machinecode gedeelte blijft na NEW aanwezig. Na NEW bent U echter regels 1 en 2 verloren en zal regel 0 alleen nog aenwezig zijn. Om regels 1 en 2 te behouden kunt U ook DELETE 3 (of het eerst te verwijderen regelnummer) TO geven.

Als U bijvoorbeeld gebruik maakt van een discdrive of een microdrive, dan is het mogelijk om op een disc of cartridge meerdere Beta-Basic programma's te zetten die gebruik maken van een en dezelfde BB code. In Uw programma (of als directe instructie) kunt u dan het volgende gebruiken:

```
CLEAR 55400 of 55800 voor BB 1.8
LOAD *"m";1;"BB" CODE
RANDOMIZE USR 58419
```

U dient er echter op te letten dat de "kale" machinecode dan aanwezig is; is dit niet het geval (bijvoorbeeld met DEF KEY's), dan zullen de genoemde CLEAR-adressen overeenkomstig aangepast moeten worden. Deze adressen krijgt u door:

```
PRINT DPEEK(23730) te geven (PRINT de ramtop)
```

U kunt nu ook enkel de machinecode SAVEn door:

```
LET RT = DPEEK(23730)
RANDOMIZE USR 59904
SAVE "BB" CODE RT + 1, 65367 - RT
```

Het kan (bij SAVEn onder andere) voorkomen dat de machinecode opnieuw gestart moet worden. U kunt dit doen door als directe instructie te geven :

```
RANDOMIZE USR  58419
```

3. OVERZICHT VAN DE KEYWORDS VAN BETA BASIC

	KEYWORDS	TOETSEN
1	ALTER	GR A
2	AUTO	GR 6
3	BREAK	BREAK/SPACE
4	CLOCK	GR C
5	DEF KEY	GR SS 1
6	DEF PROC	GR 1
7	DELETE	GR 7
8	DO	GR D
9	DPOKE	GR P
10	EDIT	O en GR SS 5
11	ELSE	GR E
12	END PROC	GR 3
13	EXIT IF	GR I
14	FILL	GR F
15	GET	GR G
16	JOIN	GR SS 6
17	KEYIN	GR SS 4
18	KEYWORDS	GR 8
19	LIST	LIST
20	LLIST	LLIST
21	LOOP	GR L
22	NEW	NEW
23	ON	GR O
24	ON ERROR	GR N
25	PLOT (een string)	PLOT
26	POKE (een string)	POKE
27	POP	GR Q
28	PROC	GR 2
29	RENUM	GR 4
30	ROLL	GR R
31	SCROLL	GR S
32	SORT	GR M
33	SPLIT	<>
34	TRACE	GR T
35	UNTIL	GR K
36	USING	GR U
37	WHILE	GR J

ALTER <attribuut-omschrijving> TO attribuut-omschrijving

De instruktie ALTER maakt een uitgebreide manipulatie van de schermattributen mogelijk. Tot die attributen behoren: PAPER, INK, BRIGHT en FLASH - die aanwezig zijn voor iedere karakterpositie. In de eenvoudige vormen kunnen we de INK- of PAPERkleur alleen wijzigen, terwijl alles op het scherm blijft :

```
10  PRINT AT 10,10;" TEST " : PAUSE 100
20  ALTER TO INK 7
```

U ziet de tekst verdwijnen, doch verloren is deze niet ! ALTER TO INK 0 zal de tekst weer tevoorschijn halen.
Uitgebreide printinstrukties kunnen onzichtbaar verlopen en dan in een flits op het scherm verschijnen.

```
 5  PAPER 7
10  PRINT INK 7; STRING$ (704,"*")
20  PAUSE 100
30  ALTER TO PAPER 1
```

Regel 30 zal alleen de papierkleur omzetten tot blauw.

Het is echter mogelijk bij een zeer ingewikkelde tekening slechts bepaalde delen van het beeld qua kleur te wijzigen. Indien voor een deel van het scherm PAPER 1, INK 6, FLASH 1, BRIGHT 1 geldt, dan kan alleen dit gedeelte gewijzigd worden. De instruktie wordt dan bv.

```
  10 LET X=0
  20 DO UNTIL X=175
  30 PLOT PAPER 2; INK 6; FLASH
0; INVERSE 1;X,X;"TEST"
  40 PRINT AT 21,0; PAPER 1; INK
6; FLASH 1; BRIGHT 1;"TEST"
  50 ALTER PAPER 1, INK 6, FLASH
1, BRIGHT 1 TO PAPER 4, INK 0,
FLASH 0, BRIGHT 1
  60 LET X=X+1
  70 LOOP
```

Het is aan te raden ALTER pas te verwerken indien het programma foutloos loopt, u kan anders onzichtbare of onleesbare listings veroorzaken of misschien onzichtbare zaken PRINTEN.

U kan ook het volgend attribuutgrapje proberen. In een lus POKEn we ATTRP :

```
80 FOR F=255 TO 7 STEP-1: POKE
23693,F: PRINT AT 0,0;"IS DIT
GOED ?": NEXT F
```

3.1 ALTER VOORBEELDPROGRAMMA'S

PROGRAMMA 1:

```
  10 PROC ATTRP
  20 DEF PROC ATTRP. CLS
  30 FOR F=80 TO 120 STEP 8
  40 POKE 23693,F: CLS
  50 LET B$=BIN$(PEEK 23693)
  60 PRINT '"  ATTRP 23693,";PEE
K 23693'',"DEC ",PEEK 23693',"HE
X$ ";HEX$(PEEK 23693),,"BIN$() "
;BIN$(PEEK 23693):
  70 PRINT AT  9,16;" FLASH ";TA
B 28,AND(128,PEEK 23693)
  80 PRINT AT 10,16;" BRIGHT ";T
AB 28;AND(64,PEEK 23693)
  90 PRINT AT 12,16;" PAPER ";TA
B 28;AND(32,PEEK 23693)
 100 PRINT AT 15,16;" INK ";TAB
28;AND(7,PEEK 23693)
 110 PRINT AT 7,2;"BIT   ";PEEK 2
3693;TAB 20;"AND()": FOR B=1 TO
LEN B$: PRINT AT 8+B,3;-1*(B-B);
"     ";B$(B): NEXT B
 120 PLOT 80,41: DRAW 0,20: PLOT
 80,65: DRAW 0,20
 130 PAUSE 120  NEXT F: END PROC
```

PROGRAMMA 2:

```
1100 DO
1110 PRINT AT 0,0;STRING$(704,"*
")
1120 PAUSE 10
1130 ALTER TO PAPER 1
1140 PAUSE 10. LOOP
1141
1142
1143
1150 DEF PROC ATTRP
1160 FOR F=254 TO 2 STEP -2
1170 POKE 23693,F
1180 PRINT "ATTRT - PEEK 23693:"
;STR$ PEEK 23693;STRING$(32,"-")
1190 NEXT F
1200 END PROC
1210 REM
1220 REM ATTRT GRAP
1230 FOR F=255 TO 7 STEP -1: POK
E 23693,F  PRINT AT 0,0,"IS DIT
GOED": NEXT F
```

AUTO < lijnnummer x > <,stapwaarde y >

De opdracht AUTO (ENTER) schakelt de automatische lijnnummering een. Uw Spectrum zet dan een regelnummer op de invoerlijn, waar U Uw Basic schrijft. Na het ENTERen van die Basicregel zet Uw Spectrum een nieuw lijnnummer onderaan het scherm. Er zijn weer verschillende formaten mogelijk. Gemakshalve geven we ze allemaal.

AUTO	Start de automatische lijnnummering vanaf de huidige (cursor-) lijn + 10, met stapwaarde 10.
AUTO 100	Lijn 100 wordt als eerste regelnummer onderaan het scherm geworpen en de stapwaarde bedraagt ook hier 10.
AUTO 50,2	Zet eerst regelnummer 50, dan 52, 54, enz. onderaan het scherm.

Indien dus geen stapwaarde wordt meegegeven, dan wordt deze automatisch 10! Is lijn 50 al aanwezig, dan zal die met bovenstaande instruktie gehaal overschreven worden, indien geENTERd wordt.

Voor de beeindiging van AUTO zal men in normale omstendigheden het ingeworpen regelnummer DELETEn en daarna de STOP-instruktie aanroepen. Ook AUTO 0 en het gebruik van een lijnnummer groter dan 9984 zullen de instruktie AUTO doen stoppen.

Is AUTO in werking en wenst U een aantal Basicregels te reserveren voor later, dan DELETE U het laatst ingeworpen lijnnummer en voert U het gewenste nieuwe en hoger liggende regelnummer in met de daarbij behorende Basic. Het dan volgende lijnnummer wordt gelijk aan het laatst ingevoerde + de stepwaarde. Zo is het eveneens mogelijk een vergeten lijn in te voegen zonder de funktie AUTO te stoppen, door er overheen te stappen. DELETE het lijnnummer en ga met cursor 7 neer lager gelegen lijnen. Wees voorzichtig met ENTER,U zou programmalijnen kunnen overschrijven of wissen.

3.3 BREAK (SHIFT + SPACE/BREAK) BREAK voor Spectrum+

In Beta-Basic 1.0 is BREAK - in feite geen keyword - zodanig aangepast dat wanneer U een van de CLOCK-instructies gebruikt de BREAK-toets ten alle tijde doet wat er gedaan moet worden, nemelijk het programma, maar ook machinecode-routines stoppen.

Indien bijvoorbeeld door verkeerd gebruik van ON ERROR het programma in een eindeloze lus geraakt is, zal de verbeterde BREAK (na circa een seconde) in werking treden. We merken op dat ook een BREAK van o.m. een FILL-instruktie enige tijd vraagt.

Vanaf Beta-Basic 1.8 wordt deze BREAK-mogelijkheid onmiddellijk na het laden van het programma aangezet.

CLOCK n

Beta-Basic bevat een digitale klok, welke gecontroleerd wordt met het keyword CLOCK. CLOCK controleert de ekties van de interrupt-gedreven 24-uurs klok met zijn 8 (=2*2*2) voornaamste formaten : CLOCK n - waarbij 0 <= n AND n <=7 . De mode n is de hoofdvariabele van de klok en zij bepaalt de stand van de mogelijke klokopties:

KLOKMODE

KLOKDISPLAY	(klok op scherm rechts boven)	ja / neen
ALARMBEEP	(BEEP hoorbaar om X uur)	ja / neen
ALARMGOSUB	(naar routine op regel Y, om X uur)	ja / neen

ALARMGOSUBLIJN Y
ALARMTIJD X

OVERZICHT VAN DE CLOCKSETTING.

MODE	ALARM GOSUB	HOORBAAR ALARM	DISPLAY
0	NEEN	UIY	UIT
1	NEEN	UIT	AAN
2	NEEN	AAN	UIT
3	NEEN	AAN	AAN
4	JA	UIT	UIT
5	JA	UIT	AAN
6	JA	AAN	UIT
7	JA	AAN	AAN

Om de klok te zien lopen op het beeldscherm , d.i. het stringresultaat van CLOCK op het scherm af te beelden in de rechter bovenhoek, gebruiken we onder meer de instruktie CLOCK 1, maar ook CLOCK 3, ...5, en ..7 doen dat. Om nu de juiste tijd in te brengen geldt bijvoorbeeld de instruktie:

CLOCK "09.29.05" of CLOCK "0929" .

Het dus is niet noodzakelijk de punten of dubbele punten tussen uren, minuten en seconden aan te geven. Tevens hoeft U niet de volledige CLOCKstring in te vullen.

CLOCK "" zet evenels CLOCK "0" de tijd op "00:00:00". Deze instruktie wordt gebruikt als startsein voor de chronometer om de duur van bepaalde routines te meten. De eindtijd wordt door de funktie TIME$ later op het scharm gePRINT.

CLOCK "" is eveneens handig indien u wenst een bepaalde sessieduur aan te houden van bv. 2 uur. Ondertussen kan u dan steeds zien hoe lang u bezig bent. U kan een alarmstring opgeven die u na 2 uur waerschuwt met een BEEP.

CLOCK "" = CLOCK "0" = CLOCK "00.00.00"

Het is mogelijk een ALARMBEEP te laten horen, wanneer een bepaald tijdstip is bereikt. De instruktie CLOCK "a10" zet het alarm op 10 uur precies. Om evenwel het alarm hoorbaar te maken, moet de instruktie CLOCK 2, 3, 6 of 7 volgen, of voorafgegaan zijn. Wenst u een hoorbaar alarm, dan dient u een ' ALARMSTRING' te ENTERen . Grote of kleine a kan gebruikt worden. Het alarm blijft in werking na NEW. CLOCK "axxxx" laat het alarm horen kort na de huidige tijd.

```
CLOCK "A07.30."  =  CLOCK "a073"

10  REM A$ = DE ALARMSTRING.
20  INPUT  " ENTER alarmtijd  CLOCK ""A"" +  "; A$
30  CLOCK "A"+A$
```

Het is mogelijk een subroutine aan te roepen wanneer de alarmtijd wordt bereikt, indien de CLOCKmode 4,5,6 of 7 is (zie in de tabel) en op voorwaarde dat er een programma loopt. Dit kan een zeer eenvoudig iets zijn, bv. 10 GOTO 10 of iets ingewikkelds. Wie een alarmroutine inbouwt, kan de GOSUB-lijn van de alarmroutine eventueel onthouden met de variabele 'GOSUBLINE'.

CLOCK lijn bv. CLOCK 200

Bij het bereiken van de alarmtijd wacht CLOCK totdat het huidige statement is uitgevoerd (wat lang kan duren indien dit juist een INPUT of PAUSE is) om daarna de alarmsubroutine te doorlopen. In een programma dat volledig in machinecode is geschreven zal CLOCK nooit een ALARMGOSUB kunnen doorlopen.

Voor het aangeven van de lijn waar de ALARMGOSUB aanvangt, geldt het formaat CLOCK lijn. Daarbij moet de lijn tussen 8 en 9999 liggen (onder 8 wordt het een CLOCKmodes i.p.v. CLOCK lijn).

De CLOCK-instruktie vertoont enkele gebreken, welke inherent aan het systeem verbonden zijn. Zo loopt de tijd niet verder indien een van volgende instrukties wordt uitgevoerd: SAVE, LOAD, VERIFY, LPRINT, LLIST en CAT of CAT#.

```
DEF KEY "1" ; "HALLO"        DEF KEY in inputmode (tekst)
DEF KEY "2" : PRINT: ..      DEF KEY met statements
```

Het DEF KEY-statement laat u toe strings of statements onder te brengen in verschillende funktietoetsen. Alle cijfer- en lettertoetsen kunnen aldus bijkomend als funktietoets worden gedefinieerd. De lengte van de funktie kan erg groot zijn. Het DEF KEY-statement is krachtig en bovendien handig in gebruik.

De inhoud van de opgeroepen funktietoets wordt onmiddellijk uitgevoerd indien het instrukties betreft. Bij voorbeeld.

```
DEF KEY "3" : PRINT MEM()
```

Bij directe instructies wordt het eerste karakter na DEF KEY "3" een : (dubbele punt).

Indien het tekst is voor INPUT (of andere) dan zijn er volgende twee mogelijkheden:

A. Tijdens het RUNnen bij het geven van een stringINPUT met een funktietoets; na het indrukken van de funktietoets zal de inhoud overgenomen worden in de INPUTstring.

Voorbeeld DEF KEY in inputmode

```
  10 DEF KEY "1"; "HALLO"
  20 INPUT "DRUK OP SYMB.SHIFT +
SPACE EN 1 "; LINE A$
  30 PRINT A$
```

Bij deze mogelijkheid is het eerste karakter na DEF KEY "1" een ";".

B. De tweede mogelijkheid is weinig zinvol, omdat de tekst dan in editmode staat, zonder een voorafgaande instruktie, zodat bij ENTER een syntaxfout wordt gemeld (met een vraagteken).

```
  10 DEF KEY "1"; "HALLO"
  Als direkte opdracht: *1
```

Is het eerste karakter na DEF KEY "1" een ':' dan kan daarachter een hele reeks statements volgen.

Voor het oproepen van een funktie drukt men de toetsen (BREAK SPACE + SYMBOL SHIFT). SPECTRUM+ gebruikers toetsen eenvoudigweg (SYMBOL SHIFT + spatiebalk) in. Deze toetsencombinatie wijzigt de cursor in een oplichtende * en zet de Spectrum in funktietoetsmode.

Het is eveneens mogelijk om een ingebrachte funktietoets op te roepen in de listing met een REM-statement of PRINT-statement.

```
10 (druk BREAK/SPACE & SYMBOL SHIFT, 1) (met statements)
20 REM (druk BREAK/SPACE & SYMBOL SHIFT, 1) (met statements
   of met string)
30 PRINT (druk BREAK/SPACE & SYMBOL SHIFT, 1) (met string)
```

Bij het regelmatig testen van een procedure in ontwikkeling is het handig deze aan te roepen met bv. DEF KEY "1": PROC TEST

Een andere mogelijkheid ...

Indien u voortdurend dezelfde DATA in moet tikken in een adresbestand bij voorbeeld, dan geeft u:

```
DEF KEY "a"; "Amsterdam"
DEF KEY "d"; "Den Haag"
```

Vraagt nu het programma INPUT " Geef nu de woonplaats op ," dan drukt u gelijktijdig op SYMBOL SHIFT en SPACE waardoor * verschijnt. Druk nu a voor Amsterdam en u heeft een correcte INPUT.

DEF KEY en RAMTOP.

Bij het inbrengen van DEF KEY-toetsen zorgt Beta-Basic voor het verlagen van de RAMTOP. De definiering van de funkties kan immers ook direkt gebeuren, dus zonder lijnnummer.

```
10 PRINT DPEEK (23730)
20 DEF KEY "1", PRINT " DE RAMTOP WORDT NU VERLAAGD "
30 PRINT DPEEK (23730)
40 DEF KEY ERASE
50 PRINT DPEEK (23730)
```

U kunt dus afgewerkte routines op deze wijze achter de RAMTOP wegbergen, als ware het eigen geschreven 'machinecode'.

Beide auteurs zijn er tot op heden niet in geslaagd de inhoud van een funktie in de listing aan te roepen. Wellicht is dit mogelijk met een USR adres.

Zie eveneens de figuren x en y.

```
10 DEF PROC naam                    openen
20 stetements                       de procedure
30 END PROC                         efsluiten
40 REM rest programme
50 PROC neem                        eenroepen
```

DEF PROC staat voor DEFine PROCedure. Deze instruktie opent de definiering van een procedure (routine). DEF PROC moet als eerste instruktie in een regel voorkomen, hoewel spaties en kleur-controle codes toegestaan zijn !

Voor de naamgeving van DEF PROC hanteert men de regels welke gelden voor de namen van variebelen. Het eerste karakter is een letter, gevolgd door een string van letters of cijfers, of door $. Spaties zijn daarbij van geen betekenis en grote en kleine letters zijn equivalent. Jammer genoeg kunnen geen keywords in de procedurenaam worden geplaatst en helaas ook geen variebelen.

Een procedure kan wel eenzelfde naam als een variebele dragen, zonder dat er problemen optreden. Een string I$="JA" en een DEF PROC I$ kunnen dus in 1 programma voorkomen.

De procedure moet worden afgesloten met een geldige END PROC instruktie (toetsen GR, 3) , zoniet volgt wel de foutmelding 33,'X No END PROC'. END PROC wordt als dusdanig gebruikt; d.w.z. zonder toevoeging van de procedurenaam.

Het statement END PROC wijst naar het einde van een benoemde procedure, en het zal voorkomen dat regels binnen de procedure worden uitgevoerd, zolang deze niet door een PROC naam statement werd opgeroepen.

Loopt de procedure eenmaal, dan was het de geldige END PROC instruktie die daertoe aanleiding gaf en die daarna tevens bepaalt wanneer de pointer weer naar het hoofdprogramma moet verspringen. Een END PROC zonder DEF PROC geeft 32, 'W Missing DEF PROC'.

De procedure wordt aangeroepen met PROC naam.

```
10 DEF PROC bleck
20 INK 7: PAPER 0: BORDER 0
30 BRIGHT 1
40 END PROC
50 PROC black
```

Roep de procedure aan met PROC bleck. Om weer tot de oorspronkelijke stand te komen gebruikt U CLS#, indien U beschikt over een interface 1 of een discdrive.

Er zijn meerdere voordelen verbonden aan het gebruik van procedures. Het gebruik van GOSUB lijn - RETURN kan nu achterwege gelaten worden. Geen verwarrende GOSUB-nummers meer te

onthouden en het programma wordt minder verwarrend. Bovendien gaat het aenroepen ven een procedure sneller dan een GOSUB . Procedures stepel je als blokjes, terwijl GOTO en GOSUB het programme een elkear weven.

Een goede reed: zet de meest gebruikte procedures van Uw programme vooraan in de listing. Dit maakt het progremme sneller!

Hoewel het niet tot de etikette van Besic behoort, is het mogelijk meerdere END PROC's in een DEF PROC te pleetsen. Wenneer evenwel de programma-uitvoering over de procedure tracht te epringen, omdet ze niet werd opgeroepen door de instruktie PROC naam, den volgt de foutmelding 32, 'W Missing DEF PROC'. Er werd immers een END PROC teveel gelezen.

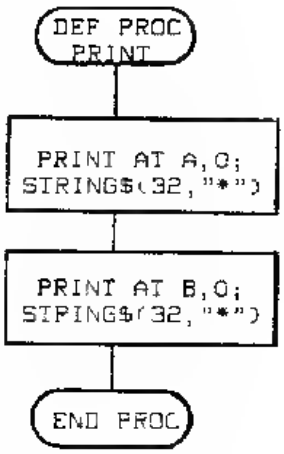
Bij gebruik van meerdere uitgengen in een procedure moet u er dus voor zorgen det deze nooit kan worden gelezen, zonder de instruktie PROC neem. Zet er een STOP voor!

```
 10 PROC TEST
 20 REM teet progremme
190 STOP
200 DEF PROC TEST
210 DO
220 stetements
230 IF INKEY$= " STOP " THEN END PROC
240 LOOP
250 END PROC
```

FLOWCHARTS VAN ENKELE PROCEDURES

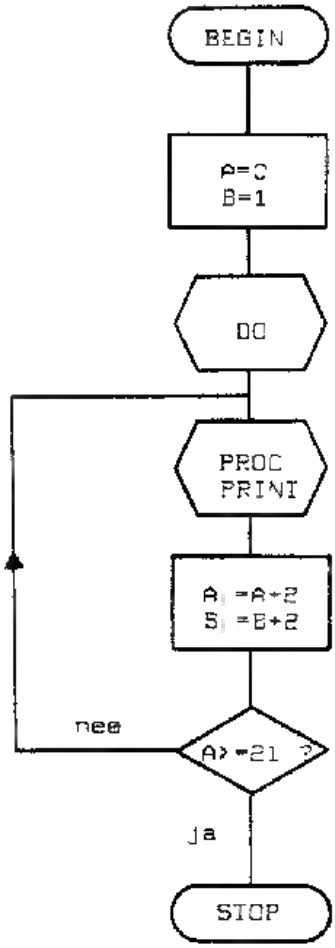
```
DEF PROC
STATEMENTS
END PROC
```

```
  10 DEF PROC PRINT
  20 PRINT AT A,0,STRING$(32,"*"
),AT B,0,STRING$(32,"#")
  30 END PROC
```

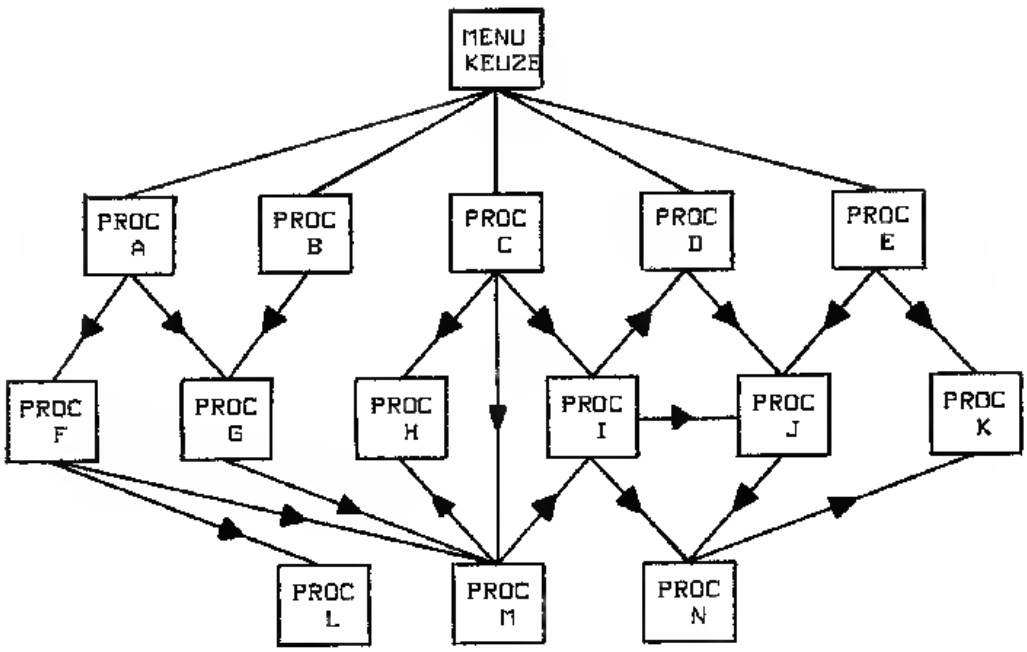


```
DO
PROC
LOOP
```

```
9000 LET A=0: LET B=1
9010 DO
9020 PROC PRINT.ZIE BOVEN
9030 LET A=A+2: LET B=B+2
9040 EXIT IF A>=21
9050 LOOP
9060 STOP
```



BLOKSTRUCTUURDIAGRAM VAN GENESTE PROCEDURES



Het is mogelijk om PROCEDURES te 'nesten' (Procedures binnen Procedures) en om vanuit meerdere procedures eenzelfde andere procedure aan te roepen.

voorbeeld.

```
 10 PROC A
 20 PROC B
 30 PROC C
 40 STOP
 50 DEF PROC A
 60 PRINT "PROC A": PRINT
 70 PROC B
 80 PROC C
 90 END PROC
130 DEF PROC B
140 DEF PROC C
150 PRINT "PROC C": PRINT
160 END PROC
170 PRINT "PROC B": PRINT
180 END PROC
```

DELETE <lijn> TO <lijn>

DELETE verwijdert alle lijnen van een bepaald blok van het Basic-programma. De mogelijke formaten en hun funkties zijn als volgt:

DELETE TO	Wist alle Basic behelve de 0-lijn, terwijl alle variabelen in het geheugen beweerd blijven.
DELETE TO 1000	Wist alle lijnen na de 0-lijn, tot en met lijn 1000.
DELETE 0 TO 0	Wist alleen de 0-lijn.
DELETE 10 TO 1000	Wist lijnen 10 en 1000 en al wat daartussen ligt.

Let wel: de opgegeven lijnnummers bij DELETE moeten in het programma aanwezig zijn, zoniet krijgt men de foutmelding 30, 'U No such line '.

Het verwijderen van de 0-lijn alleen is te gebruiken indien een met Beta-Basic ontwikkeld programma later van BB wordt ontdaen. Gebruikt u in bepaalde programma's geen Beta-Basic funkties, den kan u de 0-regel wissen om geheugen te besparen.

DELETE is binnen een programma te gebruiken op voorwaarde dat DELETE de laatste instruktie is die gewist dient te worden. Stel je voor dat bij een eerste run lijn 70 wordt gewist, terwijl 80 DELETE 70 TO 70 in het programma blijft steen. Je krijgt dan bij de tweede run de foutboodschap 'U No such line'.

Het gebruik van DELETE richt zich volledig op de basicregels die na 1 maal te zijn gelezen, niet meer nodig zijn , bv,

```
 10 DIM A$(2,5)
 20 LET A$(1)="KEUZE"
 30 LET A$(2)="MENU"
 40 DELETE TO 40
100 PRINT A$(1)+A$(2)
```

Indien de variabelen A$(1) en A$(2) beweard moeten blijven, dan kan regel 100 worden uitgevoerd , zoleng er niet wordt geCLEARd. Na CLEAR veroorzaakt regel 100 de boodschap 2, '2 Varieble not found',

In uitgebreidere zin wordt DELETE gebruikt om verschillende programma-delen na (eenmalig) gebruik te DELETEn en andere te MERGEn.

3.8 DO	CHR$ 147		GRAPHICS, D
EXIT IF	CHR$ 152		GRAPHICS, I
WHILE	CHR$ 153		GRAPHICS, J
UNTIL	CHR$ 154		GRAPHICS, K
LOOP	CHR$ 155		GRAPHICS, L

Zie ook bij de flow-chart figuren.(3)

DO : statement: statement: .. : LOOP

Het DO- en het LOOP-statement zetten samen een nieuwe vorm van lus op - vrij analoog aan een FOR-NEXT. DO wijst daarbij de start aan, welke wordt opgezocht door het bijhorende en geldige LOOP-statement.

```
10 DO : PRINT " HALLO " . PAUSE 50 : LOOP
```

De PRINTinstruktie wordt uitgevoerd totdat BREAK wordt ingedrukt.

De DO - LOOP lussen kunnen met de condities WHILE (TERWIJL) en UNTIL (TOTDAT) gespecifieerd worden, terwijl met EXIT IF de lus verleten kan worden en het programma verder zal gaan met het regelnummer na het LOOP-statement. De condities WHILE en UNTIL en het EXIT IF-statement zijn niet in een ander verband te gebruiken. Schematisch :

DO	Als LOOP wordt gelezen 'doe' den vanaf hier de instrukties tussen DO en LOOP.
DO WHILE	Doorloop de lus 'zolang' een bepaalde uitdrukking waar (1) is.
DO UNTIL	Voer de lus uit terwijl iets niet waar is (0), maar 'totdat' iets waar (1) wordt.
EXIT IF	Als iets waar is (1), spring dan naar het regelnummer na LOOP.
LOOP	Zoek de DO-start en beeindig de lus hier en ga verder met de eerste lijn na LOOP.
LOOP UNTIL	Zie bij DO UNTIL
LOOP WHILE	Zie bij DO WHILE

Het gebruik van de condities WHILE en UNTIL na het DO-statement heeft het voordeel dat indien de conditie niet waar is, de LOOP niet wordt doorlopen, maar onmiddellijk buiten de LOOP gesprongen wordt en zelfs geen data in de DO-LOOP stack worden gezet. Bij gebruik van LOOP UNTIL en LOOP WHILE wordt wel in de DO LOOP gesprongen , ook als de conditie niet opgaat, zodat dus onnodig een aantal statements wordt gelezen (uitgevoerd) en tijd verloren gaat.

EXIT IF wordt aangewend om de lus te verlaten - bij voorkeur ven ergens in het midden, eerder dan vlak bij het DO- of LOOP-stetement.

Na de IF van EXIT IF kunnen rekenkundige , relatie- en logische operators staan, net als na het gewone IF-statement.
Bij EXIT IF is het THEN-gedeelte echter vooref benoemd met een ''Ga naar de eerstvolgende lijn na het LOOP-statement.''

Bij onze tests is gebleken dat - in de meest eenvoudige vormen - een FOR NEXT-lus toch sneller is dan een DO-LOOP, terwijl een GOTO nog trager is. Nochtans maakt de DO LOOP het soms mogelijk een meer uitgebreidere FOR NEXT sterk te vereenvoudigen, zodat toch tijd en bytes gespaard kunnen worden.
Als voorbeeld voor de DO LOOP nemen we de scherm-lees routine van het ingebeelde adres-programma,dat er met een FOR-NEXT instructie als volgt uitziet:

```
  10 PRINT AT 0,0;"DEZE STRING W
ORDT OPGESLAGEN";","
  20 LET A$=""
  30 FOR F=0 TO 31
  40 IF SCRN$(0,F)="," THEN GO T
O 70
  50 LET A$=A$+SCRN$(0,F)
  60 NEXT F
  70 CLS: PAUSE 10
  80 PRINT A$
  90 STOP
```

Met de DO-LOOP:

```
  10 PRINT AT 0,0;"DEZE STRING W
ORDT OPGESLAGEN";","
  20 LET A$=""
  30 F=0
  40 DO
  50 EXIT IF SCRN$(0,F)=","
  60 LET A$=A$+SCRN$(0,F)
  70 LET F=F+1
  80 LOOP
  90 CLS: PAUSE 10
 100 PRINT A$
 110 STOP
```

De instructies tussen DO en LOOP worden herhaald totdat SCREEN$ (0,F) gelijk wordt aan een komma. Indien er gelijkenis optreedt, dan zorgt de EXIT IF instructie ervoor dat het programma (op een beschaafde manier, dus zonder GOTO) verder gaat na de LOOP.

De DO-LOOP is bijzonder handig bij submenu's onder andere:

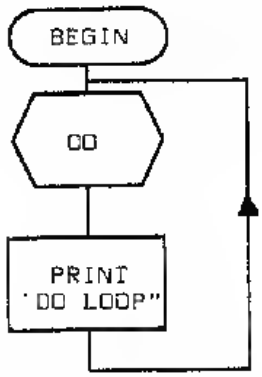
```
10 PRINT " MENU GEGEVENS"
20 DO
30 GET KEUZE
40 IF KEUZE=1 THEN INPUT "naam "; LINE a$
50 IF KEUZE=2 THEN INPUT "Woonplaats"; LINE b$
60 IF KEUZE=3 THEN INPUT "postcode "; LINE p$
70 EXIT IF KEUZE=4
80 LOOP
90 REM RESTANT
```

DO-LOOP lussen mogen op dezelfde manier als FOR-NEXT lussen genesteld zijn.
Bijvoorbeeld.

```
10 DO
20 PRINT "1e DO lus"
30 DO
40 PRINT "2e DO lus"
50 EXIT IF INKEY$= " STOP "
60 POKE 23692,255
70 LOOP
80 LOOP
```

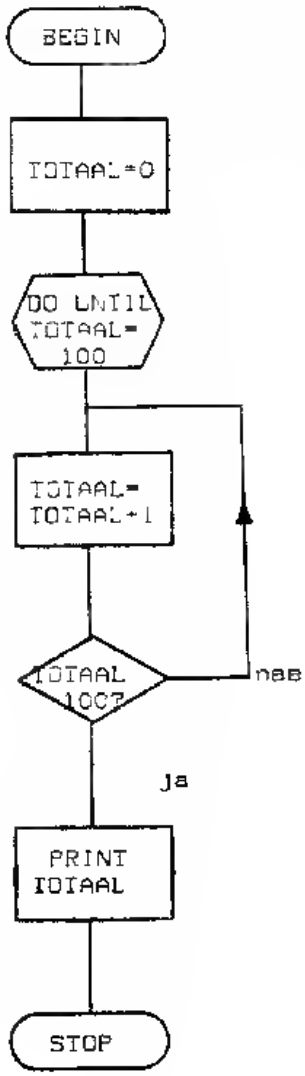
FLOW CHARTS DO .... LOOP

DO
LOOP




```
10 DO
20 PRINT "DO LOOP"
30 LOOP
```

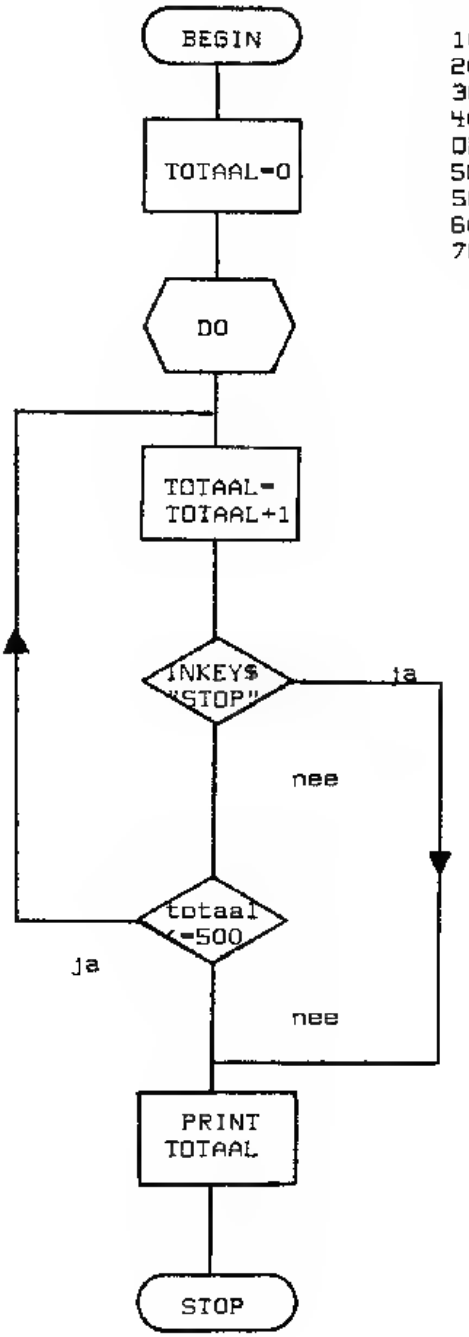
DO
UNTIL
LOOP




```
10 LET TOTAAL=0
20 DO UNTIL TOTAAL=100
30 LET TOTAAL=TOTAAL+1
40 LOOP
50 PRINT TOTAAL
60 STOP
```

FLOW CHART DO .... EXIT IF .... LOOP WHILE

```
DO
EXIT IF
LOOP WHILE
```




```
10 LET TOTAAL=0
20 DO
30 LET TOTAAL=TOTAAL+1
40 EXIT IF INKEY$=" ST
OP "
50 LOOP WHILE TOTAAL<=
500
60 PRINT TOTAAL
70 STOP
```

DPOKE adres, waarde
DPEEK adres

Dubbel POKE is in Basic equivalent met de uitdrukking:

POKE adres, waarde - INT (weerde/256)*256
POKE adres +1 , INT (waarde/256).

Deze twee regels geven aan hoe getallen tussen 0 en 65535 in machinecode worden opgeslagen. Met de DPOKE wordt dit dan : DPOKE adres, waarde.

Voor het uitlezen van de inhoud van het adres en (adres+1) om de opgeslagen waarde opnieuw te halen is er : PRINT DPEEK adres. Dit stemt overeen met volgende Basiclijn:

PRINT PEEK adres+256*PEEK (adres+1)

Dit is vooral handig omdat verschillende systeemvariabelen een waarde in een 2 byte edres onthouden.

Enkele handige tips : U kunt het aantal bytes dat door Basic alleen in beslag genomen wordt, berekenen in Besic met:

PRINT (PEEK 23627+256*PEEK 23628) - (PEEK 23635+256*PEEK 23636)

In Beta Basic doen we dat in lijn 40,

```
30 DEF PROC BASIC BYTES
40 PRINT " Aantal bytes gebruikt door Basic : ";
   DPEEK (23627) - DPEEK (23635)
50 END PROC
```

Dit komt overeen met het verschil ven de inhoud van de twee 2-byte systeemvariabelen VARS en PROG.
De uitdrukking (VARS) - (PROG) geeft dit verschil het beste weer.

VARS . de 2-byta systeemvariabele die het adres onthoudt vanaf hatwalke het Basicprogramma begint - en

PROG : de 2-byte systeemvariebele welke het edres onthoudt vanaf hetwelke de variabelen worden opgeslagen . Zie op p 121 in het SPECTRUM handboek (engelse versie).

Omdat de instrukties DPOKE en DPEEK zo dicht bij de machinetaal aanleunen, is het makkelijk deze aruit af te leiden, zeker voor korte regels als :

DPEEK (23627)-DPEEK(23635)

mcode	opcode
ED5B 535C	LD DE,(23635)
2A4B5C	LD HL,(23627)
C600	ADD A,0
ED52	SBC HL,DE
44	LD B, H
40	LD C, L
C9	RET

Op analoge wijze berekenen we het aantal bytes dat door de variabelen wordt ingenomen.

```
 95 DEF PROC VARS BYTES
100 PRINT DPEEK (23641)   -  DPEEK (23627)
110 REM  ( E LINE ) - (VARS )
120 END PROC
```

ELINE . De syteemvariabele ELINE bewaart het adres van de BASICregel die wordt ingebracht of die in EDIT mode wordt aangepast. Bij ENTERen van een nieuwe lijn wordt het edit geheugen weer gewist en neemt de inhoud van o.m. VARS EN ELINE toe.

Met DPOKE wordt machinetaal veel eenvoudiger ingebracht, wat we trechten aan te tonen met de laserontploffing van de boze spionnen. Aanvankelijk gingen we uit van een Basic-programma dat 25 decimale DATA bevatte.Deze komen overeen met machinecode. In regel 20 zijn deze nu opgenomen in 13 hexadecimale getallen. De machinecode DATA worden achter de nieuwe ramtop geDPOKEd. Zorg voor het correct invoeren van de DATA , anders kunt U na het aanroepen van de machinecode een crash verwachten.
Daal 2 van het programma, vanaf regel 120 toont u wat er allemaal gebeurt. Het derde gedeelte werpt de weggeDPOKEte DATA terug in de listing op een nieuwe regel (2000). Dit toont aan dat machinecode o.m. in hexadecimaal naar de listing kan worden omgezet.
Is het programma goed doorlopen, dan kan het geDELETEt worden. Om de laser aan te roepen is immers slechts 1 regel nodig: RANDOMIZE USR 50000. Wenst U later toch de machinecode terug te zien, gebruik dan opnieuw het derde deel nl. de KEYIN-routine van regel 200 tot en met 270.
Het SAVEn kan gebeuren met:

```
1000  SAVE "LASER"  LINE  10                (voor het programma)
1010  SAVE "LASERc" CODE 50000,26: STOP    (voor de machinecode)
```

```
  10 REM LASERSCHOT, HEX DATA
  20 DATA "0506","21C5","0300","0111","E500","B5CD",
"E103","1011","A700","52ED","F020","10C1","C9E9"
  50 CLEAR 49999
  60 LET S$=""
  70 FOR F=50000 TO 50025 STEP 2
  80 READ Z$: DPOKE F,DEC(Z$):NEXT F
  90 POKE F,201
 100 PRINT "ADRES DEC HEX PEEK DPEEK"
 130 FOR F=50000 TO 50026
 140 PRINT " "; HEX$(F);" "; USING "000";PEEK F;" ";
 HEX$(PEEK F),S$, HEX$(DPEEK(F)),"  ";
 150 IF MOD((F-50000),2)=0 THEN PRINT OVER 1;FLASH 1;
STRING$(5,CHR$ 8),S$+S$+HEX$(DPEEK(F));"<".ELSE PRINT
S$
 160 NEXT F
 170 REM LASER AAN !!!
 180 RANDOMIZE USR 50000
 190 REM DE HEX DATA TERUGHALEN
 200 LET A$="2000 DATA "+CHR$ 34
 210 LET B$="""","""
 220 FOR F=50000 TO 50025 STEP 2
 230 LET A$=A$+(HEX$(DPEEK(F))+B$
 240 NEXT F
 250 LET A$=A$( TO LEN A$-2)
 260 PRINT A$
 270 KEYIN A$
 280 REM EINDE OPHALEN
 290 DELETE 20 TO 20
```

3.10 EDIT CHR$ 13B 0-toets of GRAPHICS,(SYMBOL SHIFT +5)

EDIT < lijnnummer >

EDIT zordt voor een eenvoudig oproepen van een Basicregel. Het vervelende EDITten van Sinclair-Basic :

LIST lijn <ENTER>
BREAK
SHIFT 1 (of EDIT-toets op de plus)

is nu niet meer nodig.

EDIT krijgt men door eerst te ENTERen en dan op da 0-toets te drukkan. Het ENTERen kan soms achterwege blijven indien na de uitvoering van een opdracht of programma(deel) nog geen cursortoetsen voor verspringen in de listing werden gebruikt.

ENTER
EDIT lijn < ENTER >

Wordt geen lijnnummer aangegeven, dan EDIT men de lijn met de cursor.

In Beta-Basic 1.9 is het mogelijk een lijn te EDITten door het regelnummer ervan met een of maerdere nullen ervoor te ENTERen.

0020 < ENTER > = EDIT 20 < ENTER >

Dit kunt U met Beta-Basic 1.8 beter niet proberen, omdat Bete-Besic 1.8 deze EDIT-vorm niet kent, waardoor u de gaENTERde lijn uit hat programma zou wippen.

De oorspronkelijke lijn welke in EDITmode werd gewijzigd, kan opnieuw geEDIT worden met de Spectrum EDIT.

De cursors in EDIT-mode.

U kunt met de cursortoetsen 6 en 7 in een geEDITte lijn versneld aan het eind van deza lijn komen door:

- met cursor 6 (naar beneden) lijn per lijn naar onder te springen
- cursor 7 (naar boven) een maal in te drukken, indien de cursor op de bovenste lijn staat.

Van het eind van de regel in EDIT-mode springt u naar het begin

- met cursor 7 (naar boven) die per regel terugspringt of
- met cursor 5 (links) om pet karakterpositie terug te keren.

Da cursor zal nooit middenin een keyword kunnen terechtkomen (die zijn uit een stuk). Komt de cursor evenwal in een samengasteld funktiewoord (bijvoorbeald: MEMORY$ = FN M$) terecht dan zal het even 'uiteenvellan'. Nadat de cursor echter verpleats is, zal een en ander opnieuw in de oorspronkelijke staat zijn teruggebracht. EDIT is een directe instruktie.

ELSE statament: statemant

ELSE maakt steeds deal uit van een IF-THEN-stuktuur. In Basic-omstandigheden verapringt de programma-uitvoaring na het lezen van een IF-statement dat niet waar is, naar het volgende lijn-nummer. In Beta-Basic zal een IF-THEN-struktuur , later in de lijn gekoppald met : ELSE ... , de programme-uitvoering vervolgen met het lezen van het statement na de ELSE-instruktie, ook indien het eerste IF-statement vels wes. De dubbele punt (:) gast steeds aan het ELSE-statement vooraf!

```
10  IF ...  THEN ...  : ELSE ...
```

Het gebruik van ELSE als earste statement van een Basicregel ie fout!

Voor een schematisch overzicht van het gebruik ven ELSE, zie ook de flowcharts op de volgenda bladzijdan.

VOORBEELOEN:

```
9000 DO
9010 LET A=RNDM(1): LET B=RNDM(1
)
9020 LET C=RNDM(1)
9030 PRINT "A=";A;" B=";B;" C=";
C''
9040 IF A=B AND C=A THEN PRINT "
A=B=C": ELSE IF A=B THEN PRINT "
A=B  <>C": ELSE IF B=C THEN PRIN
T "B=C <>A": ELSE PRINT "A=C <> B
"
9050 PAUSE 0: CLS
9060 LOOP
```

VOORBEELD 2:

```
  10 DO
  20 LET A=RNDM(20)
  30 LET B=RNDM(20)
  40 PRINT AT 0,0;"A = ";A;STRIN
G$(11," ");"B = ";B
  50 PRINT
  60 IF A=B THEN PRINT "A IS GEL
IJK AAN B ": ELSE PRINT "A IS NI
ET GELIJK AAN B"
  70 PRINT
  80 IF A<B THEN PRINT "A IS KLE
INER DAN B": ELSE PRINT "A IS GR
OTER DAN B"
  90 PAUSE 0: CLS
 100 LOOP
```

```
FILL x,y  ( x en y zijn  stert-coordinaten)
FILL < INK kleur > ; x,y
FILL < PAPER kleur > ; x,y
PRINT FILLED ()
```

FILL wordt gebruikt om een bepaald gedeelte van het scherm op te vullen in PAPER- of INK-kleur, waarbij x en y de stert-coordinaten zijn. Die coordinaten volstaan om alle mogelijke vormen te vullen, gezien de FILL-instruktie zelf negaat weer er een 'rand' is.
Ofwel vult u een PAPER-gedeelte met de INKkleur van de permanente attributen of met een INKkleur die u zelf kiest.

```
FILL x,y                Vult PAPER-gedeelte met INK van ATTRP
FILL INK n;  x,y        Vult PAPER-gedeelte met INK n
```

Ook kan een INK-deel van het scherm worden opgevuld met PAPER-kleur. Tevens kan ook hier n worden weggelaten, zodat de PAPERkleur van de permanente attributen zal worden gebruikt.

```
FILL PAPER x,y          Vult INK-gedeelte met PAPER van ATTRP
FILL PAPER n, x,y       Vult INK-gedeelte met PAPER n
```

Probeert u vanop x,y een deel met INK te vullen, terwijl punt x,y reeds in INK is, dan zal er niets gebeuren. Ook "ingewikkeldere" syntaxen zijn mogelijk zoals:

```
FILL INK n; PAPER m; OVER 1; BRIGHT 1; FLASH 1; x,y
```

Hier zal met inkt n het papiergedeelte m gevuld worden, terwijl de attributen van het gevulde deel gewijzigd worden tot OVER 1; BRIGHT 1 enz.
De FILL-instruktie zal het snelst verlopen indien er veel vrij geheugen aanwezig is. FILL zet aan aantal DATA in het geheugen, waarmee wordt nagegaan of elle noodzakelijk punten werden opgevuld. Het eentel gevulde pixels kan u verkrijgen met:

```
PRINT FN FC                    PRINT FILLED ()
```

VOORBEELDPROGRAMMA 1:ROLLENDE BAL

```
10 BORDER 0: PAPER 0: INK 0
20 CIRCLE 128,88,30
30 PLOT 128,88: DRAW 30,0
40 PLOT 098,88: DRAW 30,0
50 PLOT 128,58: DRAW 0,60
60 FILL INK 1;126,90
70 FILL INK 2;126,86
80 FILL INK 3;130,86
90 FILL INK 4;130,90
100 DO
110 ALTER INK 4 TO INK 5
120 ROLL 8
130 ALTER INK 3 TO INK 4
140 ROLL 7
150 ALTER INK 2 TO INK 3
160 ROLL 5
170 ALTER INK 1 TO INK 2
180 ROLL 6
190 ALTER INK 5 TO INK 1
200 LOOP
210 REM ROLLs voor sneller
    draaiende bal
```

VOORBEELDPROGRAMMA 2: OLYMPIADE

```
 10 OVER 0
 20 DEF PROC C
 30 CIRCLE INK F;128,88,41
 40 ROLL 9,48: END PROC
 50 REM
 60 FOR F=1 TO 3: PROC C
 70 NEXT F
 80 PAUSE 50: ROLL 10,32
 90 PAUSE 50: ROLL 12,120
100 FOR F=5 TO 6
110 PROC C: NEXT F
120 ROLL 12,72: ROLL 7,16
130 OVER 1
140 FILL INK 1; 56,88
150 FILL INK 6;104,88
160 FILL INK 5;152,88
170 FILL INK 3;200,88
180 FILL INK 0;148,72
190 FILL INK 5;104,120
200 FILL INK 2;152,120
210 FILL INK 4;128,56
```

VOORBEELDPROGRAMMA 3. FILLED ()

```
 10 PLOT 60,60
 20 DRAW 0,100
 30 DRAW 100,0
 40 DRAW 0,-100
 50 DRAW -100,0
 60 FILL 70,70
 70 PRINT "AANTAL PIXELS GEVULD
:";FILLED()
```

GET getal
GET string

De GET-instruktie breidt de communicatie met het toetsenbord uit. En wel met de volgende 2 formaten:

GET a Haal of benoem het getal a via het toetsenbord.

GET a levert op de numerieke toetsen een cijfer op van 0 tot 9 en daarna op de toetsen A tot en met Z waarden van 10 tot 35 . Van a tot z worden waarden geleverd van 42 tot 67. Bij gebruik van SYMBOL SHIFT krijgt a voor verschillende toetsencombinaties dezelfde waarde.

GET A$ Haal een string via het toetsenbord.

Het benoemen van een string via het toetsenbord met GET A$ levert de string, overeenkomstig met de ingedrukte toetsen, i.e. in overeenstemming met de karaktercode. Dit betekent dat alle karakters, behalve de witte keywords , "gehaald" kunnen worden. Het gebruik van GET stopt de programma-uitvoering als bij een INPUT-statement. GET wacht niet op een antwoord in de EDIT-mode met daarna een ENTER, zoals bij INPUT. Het indrukken van 1 toets volstaat .

```
10  DO
20  GET A : PRINT A;"."
30  GET A$ : PRINT A$;"."
40  EXIT IF A$=" STOP "
50  LOOP UNTIL A= 214
60  REM 214= CODE van ENTER
70  STOP
```

3.14 JOIN CHR$ 137 GRAPHICS,(SYMBOL SHIFT +6)

JOIN <lijn>

JOIN <lijn> voegt de aangegeven lijn aan de lijn die volgt op deze <lijn> . Wordt de lijn niet gepreciseerd, dan voegt JOIN de lijn met cursor aan de eerstvolgende. In een programma met regels van 10 tot 100 en stapwaarde 10 is het dus niet mogelijk lijn 10 aan lijn 100 te koppelen met JOIN.

JOIN heeft als tegenhanger SPLIT of <> , welke de Basic-instrukties van een regel door ...:<> gescheiden met het oude lijnnummer onderaan het scherm werpt. Pas onmiddellijk dit oude lijnnummer aan het gewenste nieuwe lijnnummer of u overschrijft een regel!

Het aan elkaar plakken van verschillende regels, spaart u een belangrijke hoeveelheid bytes. Het massaal aan elkaar kleven van basicregels is nuttig indien het programma foutloos loopt en niet meer dient gewijzigd. Anders moet u weer gaan 'SPLITten'.

Het al te enthousiast JOINen van lijnen kan gevaarlijk zijn, wanneer regels worden samengevoegd die in het geheel niet mogen samengevoegd!

KEYIN string

Met KEYIN kan u via het programma basicregels aan de listing toevoegen. Met enkele regels kan de hele listing volgetoverd worden. Let op: KEYIN kan niet als directe instructie gebruikt, maar moet ten alle tijde in de listing worden opgenomen.

De string die men toekent aan de KEYIN-opdracht, vangt steeds aan met een regelnummer en een statement. We kunnen stellen dat de string een nagenoeg volledig correcte BASICregel is, met uitzondering van de vele aanhalingstekens die nodig zijn. Zie o.m. bij het laserprogramma op regel 210.
Enige voorbeelden voor de string:

```
LET A$="100  DATA"
LET A$="2000  PRINT "
```

Hierbij moet u steeds de keywords gebruiken en u tikt dus gemakshalve:

```
40 LET a$="2000": REM"
```

waarna u met de cursor twee stappen terugkeert en ": verwijdert.

PROGRAMMA-ONTWIKKELING Een KEYIN-voorbeeld.

Om verschillende strings veilig in een ontwikkelingsprogramma op te nemen, is de listing het meest geschikt, omdat men nog overzicht behoudt en correctie direkt in editmode gebeurt. De onderstaande KEYIN-procedure schrijft aan x aantal nieuwe lijnen in de listing, na dimensionering van een aantal nuttige arreys of files, welke al een naam hebben gekregen.

VOORBEELDPROGRAMMA: KEYIN

```
  10 DEF KEY "4":     PROC INKEY
  20 DEF PROC INKEY
  30 FOR F=200 TO 210
  40 LET A$=STR$ F+" LET F$(F)="
"BERICHT""
  50 LET B$=STR$ (100+F)+" LET F
$(F)=""VRAAG  """
  60 LET C$=STR$ (200+F)+" LET F
$(F)=""REPORT  """
  65 LET D$=STR$ (300+F)+" LET F
$(F)=""INPUT?  """
  70 PRINT A$'B$'C$
  80 KEYIN A$: REM een array
  90 KEYIN B$: REM een array
 100 KEYIN C$: REM een array
 110 NEXT F
 120 END PROC
 130 DEF KEY "0": DELETE 200 TO
410
```

Als mogelijkheden voor het dimensionaren van woord- of naam-arrays noemen we o.m.:

- ingeladen programma-delen
- display-titels — PRINT AT ...
- INPUTvragen in editmode
- INPUTvragen — AT ..
- INKEY$toetsen
- GETvragen
- antwoordstrings — AT .
- correctieberichten

```
DEF KEY "4": PROC INKEY ARRAYS
DEF PROC INKEY ARRAYS
```

KEYIN maakt het eveneens mogelijk om geDELETE dataregels, waarvan de inhoud nog in het geheugen aanwezig is, weer in de listing te zetten , zonder dat u ze zelf opnieuw moet intikken om bijvoorbeeld machinecode (opnieuw) in DATA regels om te zetten. Dit is uitgewerkt in het programma-voorbeeld 'Laserschot'.

Een andere mogelijkheid biedt dit voorbeeld-programma, dat eerst de DEF KEY's wist en er dan een inbrengt. Volgens de BB versie die u heeft zet u variabele rt op 55800 of 55400.

Regel 50 start een FOR-NEXT lus van RAMTOP tot aan het begin van de BB code en maakt A$ gelijk aan A$ plus de gaPEEKte karakters.

```
 5 CLEAR 55400
10 DEF KEY ERASE
20 DEF KEY "X": PRINT "defkey
inbrengen"
30 LET RT=55400: REM 55800 VOO
R BB 1.9
40 LET A$=""
50 FOR A=DPEEK(23730)+2 TO RT
60 LET A$=A$+CHR$ (PEEK A)
70 NEXT A
80 KEYIN A$
```

KEYIN op regel 80 zet A$ om in regel 2000 van het programma.

```
2000 REM "x PRINT "defkey inbrengen"
```

Indien u de volgende regels als volgt aanpast:

```
40 LET A$=""
50 FOR A=DPEEK(23730)+2 TO RT
```

Dan zal KEYIN regel 2000 als volgt creeren : en bovendien onmiddellijk uitvoeren.

```
2000 PRINT "defkey inbrengen"
```

```
KEYWORDS 1              Beta-Basic keywords
KEYWORDS 0              GRAPHICS-mode
```

KEYWORDS 1 of 0 controleert of het systeem gebruik maakt van GRAPHICS of user-defined grephics ofwel van de Beta-Basic keywords. Voor het takenen van grephics in een programma nemen we het stetement KEYWORDS 0 op, zodat de onderliggende grephics vrijkomen.

Initieel wordt KEYWORDS op 1 gezet om de Beta-Basic keywords zichtbaar te maken . Een voorbeeld:

```
500 KEYWORDS 0:     ENTER 500, EDIT 500, DELETE het
                    lijnnummer, ENTER.
                    Ons geef als directe opdracht KEYWORDS 0.

510 PRINT FLASH 1; AT 21,31;"":REM GRAPHIC 6
520 KEYWORDS 1
```

Na regel 500 geven we de directe opdracht KEYWORDS 0, zodat we het ingebrechte grafische teken onmiddellijk kunnen controleren.

Het is mogelijk lijn 520 te DELETEn , waarbij in de programme listing de BB keywords gewijzigd worden tot hun 'grephic' toetsequivalen. Deze rare listing leat toch volledig normeel verloop van het programma toe.

Omdat KEYWORDS de sleutel vormt welke toegang verscheft tot de GRAPHICS of de Beta-Basic keywords, is dit het enige BB keyword det niet kan 'afgezet' worden. Wet dan het verlies betreft van GRAPHIC 8; deze ken u vervangen door een spetie.

Wie erg veel GRAPHICS in een programma wil inbrengen, kan dit doen in Spectrum-Basic en pas later Beta-Basic MERGen.
U hoeft dan niet steeds KEYWORDS van 1 naar 0 en vice verse om te schakelen. Meer het is ook mogelijk Beta-Basic zelf ef te zetten met RANDOMIZE USR 59904, om - na het beeindigen van het GRAPHICS gedeelte van het programme - Beta-Basic weer te starten met RANDOMIZE USR 58419.

Voor een uitgebreid voorbeeld verwijzen we naar het I Tjing-programma.

```
LIST  <lijn> TO <lijn>
LLIST <lijn> TO <lijn>
```

Bij deze formaten kan het eerste of het tweede regelnummer weggelaten worden. Wordt het eerste getal niet opgegeven, dan vangt het LISTen of LLISTen aan bij de eerste Basicregel. Wordt het tweede getal niet ingevuld, dan gaat de LISTing tot aan de laatste Basicregel.

```
(L)LIST                 is equivalent met (L)LIST  TO
(L)LIST 100 TO 1000
(L)LIST 100 TO
(L)LIST  TO  1000
(L)LIST 100 TO 100
```

In het laatste voorbeeld zal slechts lijn 100 geLIST worden. Samen met TRACE vormt dit een goede basis om een programma te debuggen.

```
9000  LIST line  TO  line : PAUSE 50 : RETURN
```

Lijn 9000 zal - na de opdracht TRACE 9000 - lijn per lijn LISTen. Helaas heeft iedere geLISTe lijn een oplichtende cursor.
Tracht u een Beta-Basic programma te LLISTen, dan is het mogelijk dat de printer de keywords van Beta-Basic niet kent, waardoor een listing ontstaat met grafische tekens i.p.v. de Beta-Basic keywords. Dit geldt o.m. voor de ZX-printer, Seikosha GP-505 en Alphcom printers.
De oplossing is te vinden in het herhaald uitvoeren van de opdracht:

```
CLS: LIST lijn TO lijn : COPY
```

ENTER steeds na de opdracht COPY, want anders krijgt U de foutmelding 5, '5 Out of screen' bij het aanroepen van een instruktie welke ook gebruik maakt van het beeldscherm.

3.18 ON CHR$ 158 GRAPHICS, 0

```
GOSUB ON variabele; lijn, lijn, lijn
GOTO ON  variabele; lijn, lijn, lijn
```

ON selekteert de opgesomde lijnnummers voor een GOSUB- of GOTO-opdracht. Is de variabele 1 dan wordt het eerste lijnnummer geGOSUBd, toetst U 3 in dan wordt GOSUB 7600 uitgevoerd.

```
5000 GET keuze
5010 GOSUB ON keuze;  8000, 8200, 7600
```

Indien de keuze 4 zou bedragen , terwijl in het voorbeeld slechts 3 lijnnummers zijn aangegaven, dan wordt geen GOTO of GOSUB uitgevoerd, maar gaat het programma verder met de volgende lijn of met het volgende statement. Een nuttige aanvulling van bovenstaande lijnen:

```
5030 PRINT"UW KEUZE IS TE GROOT !":GO TO 5000
```

ON en GET zijn in het bijzonder geschikt voor menugestuurde programma's.

ON ERROR lijn

Na de executie van het commando ON ERROR lijn wordt een GOSUB uitgevoerd naar de aangegeven lijn. Er zijn 3 specifieke verlabelen welke bij ON ERROR horen en voluit getypt dienen te worden. (Grote en kleine letters zijn equivalent.)

Line en stat, die lijn en statement aangeven waar er een fout wordt veroorzaakt, en error welke de waarde van de foutboodschap aangeeft . Die waarden lopen van 1 tot 8, daarna van A=10, B=11 enz. Zoek deze waarden op in de APPENDIX over de foutboodschappen. Met uitzondering van ' 0 OK ' en '9 STOP statement zal voor alle Basic en Beta-Basic foutboodschappen een GOSUB mogelijk zijn. De onderstaande lijnen geven aan welke karakters een PRINT-fout veroorzaken.

VOORBEELDPROGRAMMA: ON ERROR

```
6000 DEF PROC TEST
6010 ON ERROR 7000
6020 FOR F=0 TO 40
6030 PRINT USING "000.  ",F,CHR$
 F
6040 NEXT F: END PROC
6999 STOP
7000 IF Error=20 AND Stat AND Li
ne=6030 THEN PRINT "PRINT ERROR
": RETURN : ELSE POP : CONTINUE
```

Indien dit niet als procedure gebruikt wordt dient men de END PROC bv. door STOP te vervangen. Anders wordt regel 6010 onterecht geGOSUBed.

We merken op dat bij hernummering (RENUM) de lijn na het stetement ON ERROR wel wordt aangepast, doch in de GOSUBlijnen van ON ERROR dient u wel de waarde van de variabele line zelf aan te passen.

PLOT x-coord, y-coord <,string>

Het is nu ook mogelijk een string op het scherm te PLOTten. De coordinaten verwijzen daerbij naar de uiterst linkse en bovenste pixel van het eerste karakter van de string. De strings hebben een maximale lengte van 32 karakters; langere strings zullen niet effectief gePLOT worden. Komt de string in de rechtse BORDER terecht, dan wordt immers de rest van de string links op het scherm op gelijke hoogte bijgePLOT. Gaat de string boven of onder de BORDER in, dan volgt uitzonderlijk de foutboodschap 11, 'B Integer out of range'. Dit wordt duidelijk met onderstaande lijnen :

```
10  LET A$= STRING$(704,"P")
20  PLOT 255,0; A$
```

CHR$ 13 is in PLOTinstruktıes een vraagteken en kan dus niet als 'carrıage return' worden aangewend!
De attribuutgegevens kunnen allen in het PLOTten van strings worden opgenomen.

Het PLOTten van strings heeft vele voordelen: we zijn niet meer gebonden aan het coordinatenstelsel van PRINT AT, zodat we met grotere precisie kunnen werken. We kunnen nu 42 karekters op het scherm plotten op 1 lijn en het aantal lijnen kan worden uitgebreid tot 25 of 26.
Voornamelijk bij grafische voorstellingen komt PLOT goed van pas. Wie meermaals titelstrings in het midden wil PLOTten, en de hoogte steeds aangeeft kan bv. een funktie definieren:

```
100  DEF FN W(W$)= 4*(32-LEN W$)
110  LET W$=" EEN TITEL"
120  PLOT FN W(W$), 160;W$
```

We kunnen ook op negatieve lijnen PLOTten, zonder een 11, 'B Integer out of range' foutmelding.

```
130 PLOT FN W$(W$),-15;W$
140 PLOT -15,-15; W$
```

VOORBEELDPROGRAMMA: 42 KARAKTERS PER LIJN PLOTTEN

```
 500 LET YOS=175
 510 LET D$=STRING$(42,"u")
 520 REM 42 karakters op 1 lijn
 530 DEF PROC chr42
 540 LET XOS=0
 550 FOR F=1 TO LEN D$
 560 IF MOD(F,42.66)=0 THEN LET
YOS=YOS-9: LET XOS=0
 570 IF YOS 10 THEN LET YOS=175
 580 PLOT OVER 1;0,0;D$(F)
 590 LET XOS=XOS+6: NEXT F
 600 LET YOS=YOS-9
 610 PAUSE LEN D$*4: END PROC
 620 FOR G=1 TO 100: PROC chr42:
NEXT G
 630 REM GEEF GO TO 1 OF RUN
```

POKE adres, string
POKE adres, waarde of instructie

De Sinclair POKE kan alleen waarden (of machinecode instructies) POKEn ; met Beta-Basic bent u nu in staat om ook strings te POKEn.
In Sinclair-Basic diant u het volgende te doen om een string te POKEn:

```
 10 CLEAR 39998
 20 LET A$=" EEN STRING "
 30 LET X=1
 40 POKE 39999,LEN A$
 50 FOR F=40000 TO 39999+LEN A$
 60 POKE F,CODE A$(X)
 70 LET X=X+1
 80 NEXT F
 90 REM om terug in te lezen
100 LET A$=""
110 FOR F=40000 TO 39999+PEEK 3
9999
120 LET A$=A$+CHR$ PEEK F
130 NEXT F
140 PRINT A$
```

In Beta-Basic :

```
 10 CLEAR 39998
 20 LET A$=" EEN STRING "
 30 POKE 39999,LEN A$
 40 POKE 40000,A$
 50 REM om terug in te lezen
 60 LET A$=MEMORY$() (40000 TO 3
9999+PEEK 39999)
 70 PRINT A$
```

Strings kunt u op verschillende geheugenplaatsen POKEN: vanaf de DISPLAY FILE (adres 16384 tot 22527), de ATTRIBUTEN (22528 tot 23295), SYSTEEM VARIABELEN + programma en variabelen (adres 23552 tot .....) en een gaCLEARd gedeelte voor DATA.

DISPLAY FILE en ATTRIBUTEN

De Display file en de attributen worden specifiek gebruikt voor opslag van schermgegevens (tekening ed.)

VOORBEELDPROGRAMMA: SCREEN POKE

```
 10 REM *** SCREEN WEGPOKEN ***
 20 FOR F=10 TO 80 STEP 10
 30 CIRCLE 128,88,F
 40 NEXT F
 50 FILL INK 7;128,88
 60 FILL INK 6;128,74
 70 FILL INK 5;128,64
 80 FILL INK 4;128,52
 90 FILL INK 3;128,42
100 FILL INK 2;128,32
110 FILL INK 1;128,22
120 FILL INK 0;128,12
130 LET Z$=MEMORY$() (16384 TO 2
3295)
140 PAUSE 100: CLS: PAUSE 200: P
OKE 16384,75
```

SYSTEEM VARIABELEN en PROGRAMMA

De systaemvariabelen en uw programa kunnen ook waggePOKEt worden. Onderstaande routine (listing 1) toont aan hoe u op aen eenvoudige manier een besic-programma kunt laten "verdwijnen" en weer terug kunt roepen door een DEF KEY. Alvorens Uw programma in te tikken dient U geheugen te reserveren door middel ven da diracte instructie CLEAR 33900. Uw programma wordt bij intikken geplaats in het gebied tussen adras 23552 (begin systaem veriabalen) en adres 33800.

LISTING 1:

```
  10 POKE 34000,MEMORY$() (23552
TO 33800)
  20 REM REST
  30 REM VAN
  40 REM PROGRAMMA
  50 REM
  60 REM
  70 REM ENZOVOORT
  80 DEF KEY "1": POKE 23552,MEM
ORY$()(34000 TO 44248)
  90 STOP
 100 NEW
```

POKE 34000, MEMORY$ () (23552 TO 33800) zorgt ar voor dat alles wet zich tussen adressan 23552 an 33800 opgeslagan wordt venaf adres 34000.

NEW zorgt er voor dat alles, schijnbaar voorgoed, verdwijnt.

POKE 23552, MEMORY$ () (34000 TO 44248) brengt achter het weggaPOKEte programma terug naar beginadres 23552, wearna het programma verder gaat wear hat gebleven wes. Het is achter niet mogelijk om twee of meerdere programma's te "koppelen" (door bijvoorbaald de regelnummering zodanig te organiseran), omdat u mat twae of meerdere systaemvariabelen zit.

Bijvoorbaald:

Programma 1 plaataan op adres 34000 tot 39248 en
programma 2 plaetsen op adras 39250 tot 44500.

Tik listing 2 in en geef RUN en daarne NEW. Controlear of het programma ward opgeslagan onder da DEF KEY 1 toets. Indien dit het geval is dan kunt U nogmaels NEW gaven en listing 3 intikken (of listing hernummaren en aanpassan; snellar !) Geef nu GOTO 9000 en vervolgens NEW.

LISTING 2

```
  10 POKE 34000,MEMORY$()(23552
TO 28800)
  20 CLEAR 33900
  30 REM REST
  40 REM VAN
  50 REM PROGRAMMA
  60 REM
  70 REM
  80 REM ENZOVOORT
  90 DEF KEY "1": POKE 23552,MEM
ORY$()(34000 TO 39248)
 100 STOP
 110 NEW
```

LISTING 3

```
  10 POKE 39250,MEMORY$()(23552
TO 28800
  20 CLEAR 33900
  30 REM REST
  40 REM VAN
  50 REM PROGRAMMA
  60 REM
  70 REM
  80 REM ENZOVOORT
  90 REM POKE 23552,MEMORY$()(39
250 TO 44500)
 100 STOP
 110 NEW
```

Als alles goed is gegaan heeft U nu twee programmeetjes in het interne geheugen opgeslagen !

Het programme van listing 2 is nog steeds opgeslagen onder de DEF KEY 1 toets, overtuig U en geef vervolgens NEW om het tweede programmeetje terug te halen.

Na de NEW geeft U als direct instructie:

```
POKE 23552, MEMORY$ () (39250 TO 44500)
```

en als wederom alles goed gegaan is heeft U thans listing 3 terug op het scherm.

DATA

Een deel van het geheugen CLEARen om strings (data in feite) op te slaan vereist enige organisatie vooraf, vooral wanneer meerdere strings (of aan heel bestand) worden gePOKEd. Enkele voorbeeld-programma's in dit boek geven hier een duidelijke demonstratie van.

POP < numerieke variabele >

POP wordt onder andere gebruikt bij ON ERROR om het terugkeeradres te verwijderen. Een andere mogelijkhaid is om da terugkeeradressen ven PROC, DO-LOOP en GOSUB te verwijderen ven de stack.

Het volgende voorbeald toont u hoe een procedure efgewerkt wordt met en zonder POP.

VOORBEELDPROGRAMMA: POP

```
  10 LET F=1: PROC test
  20 PROC test
  30 LET F=0: PROC test
  40 PROC tast
  50 DEF PROC test
  60 IF F THEN POP regel: PRINT
"proc aangeroapan op regalnummer
 ";regel;" f = 1 ! "
  70 IF NOT F THEN PRINT "f = 0!
"
  80 PRINT "procedura test",,,,
  90 IF F THEN GO TO regel+1: EL
SE END PROC
 100 STOP
```

Hieruit blijkt det als F = 1 de END PROC komt te vervallen en er dus uitgesprongen ken worden door GO TO REGEL + 1. De variabele REGEL wordt benoemd door POP en heeft de eerste keer de weerde 10 en de tweede keer 20, zodat dus een GO TO REGEL + 1 (of een of endere variebele !) dient gegeven te worden.

Als u regel 90 verandert in 90 END PROC krijgt U (wanneer F = 1) foutboodschap 32, 'W Missing DEF PROC, 90:1'.

Bij gebruik van POP als er geen data op de steck staat krijgt u foutboodschap 31 ,' U No POP data'. Dit wil zaggan dat u POP gebruikt heeft zonder dat er een GOSUB, DO-LOOP of PROC een te pes is gekomen.

```
10 PRINT
20 POP      : REM FOUT
30 STOP


10 DO
20 REM PROGRAMMA
30 REM
40 POP
50 LOOP     : REM FOUT
60 STOP
```

Verwijder regel 50 en RUN het nogmeels, zodet u goed ziet wat ar eigenlijk gebeurt.

RENUM<(beginlijn TO eindlijn)><LINE nieuwe startlijn x ><STEP y>

In de uitgeve 1.8 is hernummeren van het programma mogelijk in onderstaande formaten, hoewel er enige restricties zijn.

RENUM	Hernummert het gehele programma met lijn 10 als nieuwe aanvangslijn en stapwaarde 10.
RENUM LINE x	Hernummert het gehele programma met x als nieuwe aanvangslijn en stapwaarde 10.
RENUM LINE x STEP y	Het ganse programma met nieuwe aanvangslijn x en stapwaarde y.
RENUM STEP y	Hernummert heel het programma met 10 als nieuwe aanvangslijn en stapwaarde y.

Voor het hernummeren van een deel van het programma gebruikt U de aanwijzing (lijn TO lijn). Na de optie (lijn TO lijn) moet steeds LINE x aangegeven, zoniet ontstaan weinig zinvolle RENUM-formaten. De afwezigheid van x wijst immers lijn 10 als startlijn aan, waardoor de formaten nog slechts geldig zijn indien de eerste lijn van (lijn TO lijn) gelijk is aan 10. Is dit niet het gevel dan volgt 16, 'G No room for line'.

Blokhernummering.

```
RENUM (lijn TO ) LINE x
RENUM (lijn TO  ) LINE  x  STEP y

RENUM (lijn TO lijn ) LINE x
RENUM (lijn TO lijn ) LINE x STEP y
```

Indien een blok bij hernummering een deel van het programma tracht te overschrijven dan wordt de hernummering gestopt met de foutmelding 16, ' G NO room for line'. Deze foutmelding na RENUM verschilt in betekenis van de gelijkluidende G-foutmelding van Sinclair-Basic.

Hoewel er enige beperkingen zijn aan RENUM van Beta-Basic 1.8, is RENUM een zeer krachtige instruktie en zijn de beperkingen ervan niet zo ernstig, doch belangrijk om te kennen om aldus geen blunders te slaan bij hernummeren. De restrikties zijn de volgende:

Bij het gebruik van de keywords GOTO, GOSUB, LIST, LLIST, TRACE, ON ERROR en DELETE

- mogen er geen VAL"n" voorkomen,zoals in GOTO VAL"10",
- zijn variabelen die een lijn aangeven als in GOTO of GOSUB 10*a of (L)LIST A TO A+50 niet toegestaan.

Beta-Basic 1.9 vergemakkelijkt de syntax en breidt de mogelijkheden van de hernummering nog uit.

RENUM lijn TO lijn <LINE x> <STEP y>

Dit veroorzaakt een 'blokhernummering' zoals in Beta-Basic 1.8, maar nu dienen geen haakjes meer aangegeven.

RENUM lijn LINE x

Hernummert of verplaatst een regel.

RENUM * lijn LINE x

Is hetzelfde als EDIT lijn en aanpassen van de lijn.

RENUM * lijn TO lijn <LINE x> <STEP y>

Met sterretje!
De 'Blockcopy' maakt van het aangegeven blok copieen en plaatst deze vanaf lijn x met een stapwaarde y verspringend.

Bovendien worden rekenkundige uitdrukkingen als GOTO a*20 + 100, evenaans hernummerd; een logische uitdrukking als GOTO VAL"100" wordt evenwel niet hernummerd! Gebruik dus misschien liever geen VAL in een GOTO- of GOSUB-opdracht, indien er nog regelmatig hernummerd moet worden.

Kunnen niet alle regels naar behoren hernummerd worden, dan geeft versie 1.9 nu een opgave van de regels die nog manueel aangepast moeten worden, onder de hoofding:

Failed at : lijn, statement.

Gebruikt u toch GOTO en GOSUB in combinatie met moeilijker te berekenen lijnnummers, of met VAL, schrijf dan vooral vooraf op een stuk papier welke GOTO's waar neartoe springen.

De hernummering kan ook plaats vinden over regels heen, gezien bijvoorbeeld:

RENUM 10 TO 30 LINE 70

10	...		40
20	...	wordt	50
30	...		60
40			70 ...
50			80 ...
60			90 ...

de regels 10 tot 30 over de regels 40, 50 an 60 heen hernummert, vanaf regel 70.

In LIST EN LLIST struikelt RENUM 1.9 wel over het line TO line stetement mat de foutboodschap "Failed at : " doch, in dit geval dient toch niets gecorrigeerd. Dit gebeurt eveneens bij GOSUB ON var; lijn, lijn.

```
ROLL richting;< x coord, y coord; breedte, hoogte>
```

Met Beta-Basic 1.0 is het nu mogelijk de gehele display of een deel daarvan te bewegen in de richtingen van de cursor. De eenvoudigste ROLL en SCROLL-instrukties verplaatsen het beeld met 1 pixel. Deze 4 eenvoudigste (SC)ROLL-instrukties zijn :

```
(SC)ROLL richting
```

(SC)ROLL 4	: naar links	(SC)ROLL 8	: naar rechts
(SC)ROLL 5	: naar onder	(SC)ROLL 7	: naar boven

Gezien hier slechts per pixel wordt verschoven zal voor een trage beweging van de gehele display een van bovengenoemde instrukties in een lus gebracht worden.

```
FOR F = 1 TO 176 : ROLL 7 : NEXT F
```

Deze lus zal langzaam de display 1 meel opwaarts ''rond'' ROLLen! Dit kan ook snel indien de instruktie (SR)ROLL richting ,n wordt gebruikt. De display beweegt dan met n eantel pixels in een keer, waardoor het echte rollend effect afneemt of verdwijnt.

```
(SC)ROLL richting, aantal pixels

(SC)ROLL 7, 40          beweegt in 1 snelle beweging 5
                        lijnen van de display. Om
                        dit in stappen te doen :
FOR F= 1 TO 40 STEP 8
(SC)ROLL 7, 8 : NEXT F
```

Wordt een deel van de display in de BORDER verschoven met een ROLL-opdrecht, dan komt dit deel weer tevoorschijn aan de tegenoverliggende zijde. Wanneer een deel van de display in de BORDER wordt geSCROLLd, zal dit deel van de display verloren gaan! Dit is het grote verschil tussen ROLL en SCROLL!

De moellijkere ROLL- en SCROLL-formaten (SC)ROLLen alleen een deel van de totale display.

```
ROLL richting ; <x-coord, y-coord; x-breedte, y-hoogte>
```

Hierbij wijzen de formaat-onderdelen het volgende aan :

x- en y-coord:	uiterst linkse,'bovenste' pixel; het aanvangspunt
x-breedte:	de breedte van het te (SC)ROLLen schermdeel: de raambreedte, uitgedrukt in aantal karakters (van 1 tot max. 32)
y-hoogte:	raamhoogte van 1 tot 176 (van boven naar beneden gezien)

```
8000 FOR F =200 TO 255 : PRINT CHR$ F;:NEXT F
8010 FOR G = 1 TO 88 : SCROLL 7; 0,175; 32,100 : NEXT G
```

Beta-Basic 1.8 breidt de mogelijkheden van (SC)ROLL nog uit. Het moeilijkare formaat is nu :

(SC)ROLL richting,<,n pix>;<x-coord,y-coord;x-breedte,y-hoogte>

Het aantal pixels dat een gegeven deel van de display in een keer moet verschuiven, kan nu gespecifieerd. Doet men dit niet dan wordt n=1. De ware verbatering bestaat in het gebruik van cursorgetallen van 1 tot 12, met volgende betekenis :

Cursorgebruik in (SC)ROLL -opdrechten.

CURSORS: 1 tot 4 verpleetaing van de attributen alleen,
5 tot 6 verpleetaen aileen de diaplay alleen,
7 tot 12 bewegen attributen en display.

Omdat de attributen worden toegekend aan een 8*8 pixel-vierkent, is het elleen mogelijk da attributen te varplaatsen met 1 karakterpositie. Het is daarom aan te raden bij de verpleetsing ven attributen een waarde 8 te henteren voor variabele n.
Tevens is het nodig dat strings welke gePLOT worden en die verschillen in hun tijdelijke ettributen met de parmanenta, det zij op een PRINT AT plaats worden gePLOT. Zoniet worden de tijdelijke ettributen vardeeld over 2 lijnen, waarbij een grotere PAPERkadar ontsteet rondom de string. Anderzijds is dit zeer handig in titels of tekeningen!

3.25 SCROLL CHR$ 162 GRAPHICS, S

Zie ook ROLL

SCROLL heeft dezelfde formeten als ROLL, met 1 uitzondering: SCROLL is ook mogelijk zonder enige aanvulling:

SCROLL verpleetst het beeldscherm met 8 pixels.

SCROLL doet echter het gedeelte dat in de BORDER is verschoven vardwijnen. ROLL doet dit niet!
Omdat de formaten reeds verklaard wardan bij ROLL, geven we hieronder enkele nuttige voorbeeldan. In het eerste voorbeeld wordt het scherm verdeeld in 2 ramen of venaters.

VOORBEELDPROGRAMMA: SCROLL

```
  10 LET D$=" DIT IS HET EERSTE
RAAM "
  20 LET E$=" DIT IS HET TWEEDE
RAAM "
  25 PRINT AT 11,0;STRING$(32,"-
")
  30 PRINT PAPER 6;AT 0,0;STRING
$(352," ")
  40 FOR R=1 TO 40
  50 PRINT AT 21,0;E$;AT 10,0; P
APER 6;D$
  60 PAUSE 30
  70 FOR H=1 TO 1: SCROLL 11,8;0
,175;32,88: NEXT H
  80 FOR H=1 TO 8: SCROLL 11,1;0
,80;32,81: NEXT H
  90 NEXT R
```

SORT string array of numarieke array of string

SORT sorteert of rangschikt arrays volgens grootte of volgens alfabet (d.i. volgens de grootte van karaktercode, zoals opgenomen in het Spectrum handboek).
Het ordenen gebeurt met hoge snelheid : 1/5 seconde voor 100 strings met LEN 10. De snelheid zal relatief weinig afnemen met toenamende lengte van de strings. Belangrijker voor de snelheid is het aantal strings. Beneden 200 strings duurt het ordenen ongeveer 0.7 seconde, 400 strings eisen ongeveer 3 secondan.

In zijn eenvoudige vorm ordent SORT een string, bijvoorbeeld:

```
INPUT B$ : SORT B$ : PRINT B$
```

Dit geeft als resultaat '-BBaaceist' indien de INPUT 'Beta-Besic' was. Dit is evenwel niet mogelijk met een string uit een array. Indian "Bete-Basic" de eerste string is uit de array a$ (10,10) dan geeft de opdracht:

```
SORT a$(1)
```

aan OK, maar blijkt er toch niets te zijn gewijzigd.

Heeft u een bestand met gegevens aangaande uw uitgaven dan kan u dit ordenan naargelang de grootta van uw uitgaven met

```
SORT A$(),

 5  DIM A$ (3,32)
10  a$(1)=" 50 fl    herstelling radio"
20  a$(2)="300 fl    huishuur"
30  a$(3)="120 fl    alimentatie week 34"
```

De array wordt geordend met het eerste karakter van elka string als aanvangsvergelijkingspunt en het resulteert in:

```
a$(1)=" 50 fl    herstelling radio"
a$(2)="120 fl    alimentatie week 34"
a$(3)="300 fl    huishuur"
```

Let erop dat de kleinste getallen eerst komen te staan.
Dit is niet het geval bij het ordenen van een numerieke array!

Indien u een alfabetische lijst wenst van de uitgaven, met SORT A$() (12 TO) dan wordt geordend met het 12de kerakter als eerste vergelijkingspunt tot het laatste karakter. Het resulteat:

```
SORT A$()(12 TO )

a$(1)="120 fl    alimentatie week 34"
a$(2)=" 50 fl    herstelling radio"
a$(3)="300 fl    huishuur"
```

Het is niet noodzakelijk een hele array te ordenen. Heeft u een grotere array, met verschillende onderdelen, of wenst u slechts een deal van de array te ordenen, dan kan dit met :

SORT A$ (stringnummer TO stringnummer)

SORT A$ (10 TO 30) zal van array a$ de 10de t/m de 30 string ordenen, met het eerste karaktar als uitgangspunt.
De combinatie van de twee formaten is eveneens geldig:

SORT A$ (10 TO 30) (12 TO)

Omdat SORT de karaktercode hanteert bij het ordenen, is het eveneens mogelijk een numerieke array, welke met FN C$ tot 2-karakterstrings werd omgezet, te ordenen.
Wel kan gezegd dat sorteren van getallen langer duurt dan het ordenen ven strings, gezien dient gecontroleerd of de getallen positief of negatief zijn en in welke vorm ze voorkomen.

Bij het ordenen ven een numerieke erreg worden de grotere getallen vooreen geplaatst. Er is daarom een SORT INVERSE voorzien. INVERSE sorteren ie ook mogelijk met strings.

SORT INVERSE A ()

VOORBEELDPROGRAMMA: SORT

```
 10 DIM A$(21,2)
 20 FOR F=1 TO 21
 30 LET A$(F)=STR$ F
 40 NEXT F
 50 SORT A$()
 60 DIM A(21)
 70 FOR F=1 TO 21
 80 LET A(F)=F
 90 NEXT F
100 SORT A()
110 PRINT
120 FOR F=1 TO 21
130 PRINT TAB 8;A(F)
140 NEXT F
150 SORT INVERSE A()
160 PRINT AT 0,0
170 FOR F=1 TO 21
180 PRINT A$(F),TAB 4;A(F)
190 NEXT F
```

Zie ook JOIN

SPLIT is geen werkelijk keyword. Men brengt het teken <> in na een statement (dit is na de dubbele punt). De Basicinterpretar zal <> na : steeds als direkt commendo eenvaarden. Zo is het niet mogelijk de SPLIT-opdracht in de listing op te nemen, wat trouwens weinig zinvol is. Geat een REM-stetemant de SPLIT-opdracht vooraf, den blijft het teken <> wel in de listing, maar wordt de opdracht niet uitgevoerd.

```
3040 PRINT AT 0,0; " HOOFDMENU ": <> GOTO 8900
```

Bij het ENTERen van lijn 3040 werpt de Spectrum volgende lijn in de edit-mode onderaan :

```
3040 GOTO 8900,          terwijl in de listing

3040 PRINT AT 0,0;" HOOFDMENU "
```

komt te staan. U dient dearom onmiddellijk na het ENTERen ven de te splitsen lijn het oude regelnummer aan te passen aan het gewenste nieuwe regelnummer, zoniet zult U het eerste deel van 3040 overschrijven en verliezen.

Denk vooral bij het corrigeren van langere regels aan de SPLITopdracht. Voor het copieeren ven een deel ven een regel ken men met SPLIT dit deel uit de regel afsplitsen, een copie dearvan maken en het afgesplitste deel daarna weer bij het oude voegen met JOIN. Tracht evenwel te voorkomen det een programma vele equivalente lijnen bevat. De bezitters ven Beta-Basic 1.9 kunnen regels copieren met RENUM * n TO m LINE l

```
TRACE <lijn>      Start het TRACEn
TRACE 0           Stopt het TRACEn
```

TRACE maakt eenvoudig debuggen mogelijk, omdat TRACE <lijn> een GOSUB uitvoert naar de aangegeven lijn na de executie van ieder statement. Meest nuttig zijn nu de twee variabelen - line en stat - die in de TRACElijn verwerkt kunnen worden, om de huidige lijn en het daarin gelezen statement op het scherm te PRINTen. Ook LIST line TO line zal van elke regel een list geven, wat al meer naar debuggen toegeat.

```
   10 TRACE 9000
   20 REM of direkte opdracht
  100 PROGRAMMA
 8999 STOP
 9000 PRINT AT 21,0; "Lijn: ";line
 ,"statement: ";stat: PAUSE 33:  R
 ETURN
```

Het TRACEn wordt gestart met TRACE 9000, en gestopt met TRACE 0, dit met een direkte opdracht of in een programmaregel. Ook RUN en CLEAR zetten het TRACEn af.
PLOTten maakt de TRACE-GOSUB niet sneller:

```
 8999 STOP
 9000 PLOT 0,7;STR$ line+"  "+STR
 $ stat: PAUSE 7: RETURN
 9010 PRINT #0;AT 0,0;"lijn: ";li
 ne,"statament: ";stat: LIST #0;l
 ine TO line: PRINT #0;STRING$(32
 ," "); PAUSE 7: RETURN
```

Het is best mogelijk meerdere TRACElijnen in 1 programma te verwerken. We kunnen zelfs 'TRACE lijn' en 'TRACE 0' ook beschouwen als een alternetieve mogelijkheid om routines op te roepen. Maar dit is evenwel een weinig aantrekkelijke mogelijkheid.

```
   10  TRACE 9030
   30  TRACE 0
 8999  STOP
 9030  FOR F= 1 TO 10
 9040  PRINT CHR$ (F+200)
 9050  NEXT F: PAUSE 0: RETURN
```

USING formaatstring, getal
USING$ (formaatstring, getal)

USING en FN U$ geven vorm aan de te PRINTen getallen .
Het equivalent in Basic kan zijn :

```
190  INPUT"GETAL";GETAL
200  LET GETAL = INT (100*GETAL + .5) / 100
210  LET G$= STR$ GETAL
220  LET F= 1
225  DO
230  EXIT IF G$(F) = "."
240  LET F=F+1
245  LOOP
250  PRINT AT 21, 16-F; G$
260  FOR F= 1 TO 8: ROLL 7 : NEXT F
270  GOTO 190
```

Men bepaalt zelf het aantal cijfers na of voor de komma.
De formaatstring van USING beschikt over de gegevens:

```
#  : een spatie
0  : voorafgaande 0
```

en 0 kunnen beide aangewend worden om de vorm na de komma aan te geven.

```
PRINT USING "##.#";12.3456
```

De formeetstring kan als volgt opgesteld worden (met het getal 12.3456):

```
"##.#"       geeft      12.3
"###.#"                 12.3
"####.##"               12.35
"000.00"               012.35
"00"                    12
"'00.00"               '12.35
"0.00"                 %..3
"fl 00.0"           fl 12.3
"Bfr 00.0"         Bfr 12.3
```

De laatste voorbeelden tonen duidelijk dat vooraan in de formaatstring allerlei andere karakters kunnen gepleatst worden. Het voorbeeld met het resultaat %..3 is een geval van overflow.

USING kan enkel gebruikt wordan in PRINT-opdrachtan en is ook mogelijk met wetenschappalijke notatie .

```
INPUT "GETAL ";X
PRINT USING "0000.00"; GE3      REM  GE3 = G*1000
PRINT USING "####.##"; G*2
```

De funktie USING$ is analoog met het keyword USING, meer kan ook gebruikt worden in alle instrukties die normaal ook strings manipuleren, zoals LET.

VOORBEELDPROGRAMMA'S: USING

```
10 DIM B$(4,11)
20 LET B$(1)="FL  0000.00"
30 LET B$(2)="BFR 0000,00"
40 LET B$(3)="$   0000.00"
50 LET B$(4)="'   0000,00"
60 FOR F=1 TO 4
70 LET BEDRAG=RND*150
80 LET A$=USING$(B$(F),BEDRAG)
: PRINT A$
90 PRINT USING B$(F);BEDRAG
100 NEXT F
```

VOORBEELD 2:

```
10 PRINT AT 0,0;"MET USING
ZONDER USING "
20 FOR F=1 TO 10
30 LET GETAL1=RNDM(10000)/,37
40 LET GETAL2=RNDM(100)/,3765
50 PRINT USING "00000,00";GETA
L1;STRING$(8," ");
60 PRINT GETAL1: REM ZONDER
70 PRINT USING "00000.00";GETA
L2;STRING$(8," ");
80 PRINT GETAL2: REM ZONDER
90 NEXT F
```

N.B. Een FORMAAT dat ook mogelijk is:

```
PRINT USING "FAKTUURBEDRAG   FL ########.##";F8
```

4. FUNKTIES VAN BETA-BASIC

	NAAM	TOETSEN	VERSIE	KORTE OMSCHRIJVING
1	AND	FN A(	1.8	'Bit-na-bit' AND van 2 getallen
2	BIN$	FN B$	1.8	Decimaal naar binair conversie
3	CHAR$	FN C$	1.0	Getal naar 2-karaktersting conversie
4	COSE	FN C(	1.8	Cosinus sneller, 4 cijfers na komma
5	DEC	FN D(	1.0	Hexadecimaal naar decimaal conversie
6	DPEEK	FN P(	1.0	Dubbele peek
7	EOF	FN E(	1.9	Wijst het einde van een file aan
8	FILLED	FN F(	1.8	Geeft aantal gevulde pixels
9	HEX$	FN H$	1.0	Decimaal naar hexadecimaal conversie
10	INSTRING	FN I(	1.0	Versnelde opzoekmogelijkheid
11	MEM	FN M(	1.0	Vrij geheugen
12	MEMORY$	FN M$	1.8	Zet geheugen(deel) in een string
13	MOD	FN V(	1.8	Geeft rest na deling 2 getallen
14	NUMBER	FN N(	1.0	Zet string van FN C$ weer in decimaal
15	OR	FN O(	1.8	'Bit-na-bit' OR van 2 getallen
16	RNDM	FN R(	1.8	Idem RND, maar sneller
17	SCRN$	FN K$	1.8	SCREEN$ met herkenning van UDG's
18	SINE	FN S(	1.8	Sinus sneller, 4 cijfers na komma
19	STRING$	FN S$	1.0	Herhaald PRINTen van string
20	TIME$	FN T$	1.0	Geeft de CLOCKstring op een moment
21	USING$	FN U$	1.0	Formattaart te PRINTen getallen
22	XOR	FN X(	1.8	'Bit-na-bit' XOR van 2 getallen

4.1 AND (getal,getal) FN A (

De AND van Beta-Basic verschilt van de AND van Spectrum-Basic in zijn syntax. AND van de Spectrum-Basic koppelt logische en/of rekenkundige bewerkingen aan elkaar. De Beta-Basic AND levert een 'bit-na-bit' AND van de 2 opgegeven getallen, welke overigens tussen 0 en 65535 dienen te liggen. De volgorde van de twee getallen is van geen belang.
Een 'bit-na-bit' AND vergelijkt de getallen in hun binaire vorm. Het bitresultaat wordt 1 indien in beide getallen eenzelfde bit 1 is. In andere gevellen wordt het resultaat 0.

Bij voorbeeld: AND (249,89)

```
        11111001    249   F9
AND     01011001     89   59
 =      01011001     89   59
```

Dit heeft als logisch gevolg dat de instruktie AND (getal, 0) altijd gelijk is aan 0. Voorts resulteert elke AND waarin het getal 255 is opgenomen, in het tweede getal, indien dit tweede getal kleiner is dan 255. Een AND met 65535 zal altijd het tweede getal als uitkomst geven.
Het resultaat van iedere AND-opdracht moet steeds kleiner zijn dan (of gelijk zijn aan) het kleinste getal van de AND-bewerking.
VOORBEELD VOOR HET GEBRUIK:

```
  10 PRINT BRIGHT 1; PAPER 4; FL
ASH 1; INK 5;AT 10,10;"BETA BASI
C"
  20 IF AND(007,ATTR (10,10)) TH
EN PRINT "INKT KLEUR ";AND(7,ATT
R (10,10))
  30 IF AND(128,ATTR (10,10))=12
8 THEN PRINT "FLASH IS AAN": ELS
E PRINT "FLASH UIT"
  40 IF AND(64,ATTR (10,10)) THE
N PRINT "BRIGHT AAN": ELSE PRINT
 "BRIGHT UIT"
  50 PRINT INT (AND(63,ATTR (10,
10))/8;" IS DE PAPERKLEUR"
  60 STOP
```

4.2 BIN$ (getal) FN B$ (

CONVERSIE DECIMAAL NAAR BINAIR

BIN$ vertaalt decimale getallen tot binaire strings met een langte (LEN) van 8 digits voor getallen kleiner van 256 en in strings met LEN 16, indien de getallen groter dan 255 en kleiner zijn dan 65536. Te grote waarden geven 11, 'B Integer out of range' .

```
  10 DEF PROC DEC TO BIN
  20 PRINT BRIGHT 1; INVERSE 1;"
DEC () TO BIN$()
  30 DO
  40 INPUT "  Geef DECIMAAL getal
";GETAL
  50 PRINT USING$("00000",GETAL);
" BINAIR ";TAB 14;BIN$(GETAL)
  60 INPUT "NOG EEN ? J/N ";J$
  70 EXIT IF J$="N"
  80 LOOP
  90 END PROC
```

CONVERSIE DECIMAAL NAAR 2-KARAKTERSTRING

Deze funktie zet integere getallen tot 65535 om tot 2-karakter stringe. Dit kan gezien er een 256 karakters zijn en 256*256-65536. Omdat 0 er ook bijhoort maximaal tot 65535. De berakening van de code van de 2 karakters gebeurt equivalent in Basic met:

```
  10 REM IN SPECTRUM BASIC
  20 LET GETAL1=255
  30 LET GETAL2=INT (GETAL1/256)
  40 LET GETAL3=GETAL1-GETAL2*25
6
  50 LET C$=CHR$ (GETAL2)+CHR$ (
GETAL3)
  60 PRINT C$

  70 REM IN BETA BASIC
  80 PRINT CHAR$(255)
```

Voorbaaldjes:

```
PRINT CHAR$ (65535)          geeft  " COPY  COPY "
PRINT CHAR$ (256)                   "??"
PRINT CHAR$ (255)                   "? COPY "
PRINT CHAR$ (0)                     "??"
```

Indien het getal kleiner is dan 0 of groter dan 65535, dan volgt boodschap 11, 'B Integer out of range'.

Het stringresultaat zal vaak de foutboodschap 20, 'K Invalid colour' geven, indien men het tracht te PRINTen, omdat het print controle codes kan bevatten. Om zulke vreemda stringe opnieuw om te zetten tot getallen gebruik je de funktie NUMBER().

Dit allea laat een zeer ekonomische opslag van getallen toe.

```
   5 ON ERROR 100
  20 DIM Z$(100,2): LET F=1
  30 DO
  40 LET Z$(F)=CHAR$(F*500)
  50 REM ** Z$ zonder NUMBER is
niet altijd printbaar >>> ON ERR
OR
  60 PRINT F;TAB 6;NUMBER(Z$(F))
,Z$(F)
  70 LET F=F+1
  80 LOOP WHILE F<=100
  90 STOP
 100 IF error=20 AND line=60 THE
N PRINT INVERSE 1;"ERROR !": RET
URN : ELSE POP : CONTINUE
```

Geeft de cosinus asn van het aangegeven getal, maar met een geringere nauwkeurigheid den de COS van Spectrum-Basic. Toch bedraagt dit nog 4 cijfers ns de komma, wet voor vele doeleinden ksn volstasn. De berekening is ongeveer 60% sneller, wet voor bepaalde doeleinden wenselijk kan zijn. Als voorbeeld zullen we een cosinusfunktie uitPLOTten en het tijdsverschil tuseen COS en COSE negsen.

```
10  CLOCK 1: CLOCK ""
20  FOR F= 0 TO 255
30  PLOT F,88+80*COS (F/128*PI)
40  NEXT F
50  PRINT AT 21,0; TIME$ ()
60  STOP
```

Leat eerst dit programma met de gewone COS van Basic lopen, vervang daarne COS door COSE ven Beta-Basic en vergalijk de tijd. Het tijdsverschil voor de routine-uitvoering bedraagt hier 66%! 10 seconden met Spectrum- en 8 seconden met Bete-Baeic.

Om meerdere golven te verkrijgen maakt U 128 van regel 30 kleiner bv. 64 en vervangt U 80 door 40. DELETE daarna regels 10 en 50 (of zet er een REM voor) en geef op

```
60 FOR F =1 TO 1000: ROLL 5: NEXT F
```

U heeft nu een lopende funktie of golven.

Een sinusfunktie verkrijgT U door COSE door SINE te vervangen of door een kopie van 30 op 32 met SINE i.p.v. COSE.

4.5 DEC (hexedecimale etring) FN D("

CONVERSIE HEXADECIMAAL NAAR DECIMAAL

DEC ("G") geeft het decimale equivalent van het hexedecimeal getal G. De lengte van het hexsdecimale getsi meg verieren van 1 tot 4 en zowel kleine als hoofdletters worden aanvaard.

```
 50 DEF PROC HEX TO DEC
 60 LET D$=" HEX TO DEC"
 70 DO : PRINT D$
 80 INPUT " Geef de hexadecimale op!"; H$
 90 PRINT h$ ;" hex = "; DEC(h$);" dec "
100 INPUT" Nog eene?";y$
110 EXIT IF  Y$<>"Y"
120 LOOP
130 STOP: REM NAAR HOOFDMENU
140 END PROC
```

Deze functie is aanwezig in BB 1.8 en is alleen van belang wanneer u met microdrive READfiles werkt.

Het volgende voorbeeld demonsteert de werking:

```
 10 REM WEGSCHRIJVEN
 20 OPEN #4;"M";1;"NAAM"
 30 DO
 40 INPUT "NAAM ";N$
 50 EXIT IF N$="" STOP "
 60 PRINT #4;N$
 70 INPUT "ADRES ";A$: PRINT #4
;A$
 80 LOOP
 90 CLOSE #4
100 STOP
110 REM INLEZEN
120 OPEN #4;"M";1;"NAAM"
130 DO WHILE NOT EOF(4)
140 INPUT #4;N$: PRINT N$
150 PAUSE 1
160 INPUT #4;A$: PRINT A$
170 PAUSE 1
180 LOOP
190 CLOSE #4
200 STOP
```

Bij het stoppen met het wegschrijven wordt uit de lus gesprongen en de file wordt gesloten. Bij het terug inlezen blijft de waarde van EOF nul zolang de laatste DATA nog niet werd ingelezen. Na het inlezen van de laatste DATA wordt EOF gelijk aan 1 en wordt er uit de inlees-lus gesprongen. U behoeft dus niet meer Uw laatste data te merkeren met een "eind-teken" .

Omdat deze funktie niet blijkt te werken op de OPUS, geven we u het alternatief voor EOF in de tweede listing. USR 432 heeft als resulteat het aantal bytes dat nog over is in de huidige stream. Wordt USR 432 gelijk aan 0, dan is dus het einde van de file bereikt. Is de stream van ongekende lengte, dan is het resulteat gelijk aan -1.

VOORBEELDPROGRAMMA voor OPUS DISCOVERY1 zie volgende pagina.

VOORBEELDPROGRAMMA

EOF VOOR GEBRUIK MET OPUS DISCOVERY1 DISC-DRIVE.

```
  10 REM DISCOVERY VERSIE
  20 DIM V$(5,10): DIM N$(5,20)
  30 LET V$(1)="NAAM"
  40 LET V$(2)="STRAAT"
  50 LET V$(3)="CODE PLAATS"
  60 LET V$(4)="TELEFOON"
  70 LET V$(5)="INDIKATIE"
  80 REM PROCEDURES INSTELEN
  90 DEF PROC PRU: FOR G=1 TO 5:
PRINT AT G*2,0;V$(G): NEXT G:  E
ND PROC
 110 DEF PROC OUT
 120 OPEN #4;"M";1;"BESTAND" OUT
 130 PROC UIT: END PROC
 150 DEF PROC UIT
 160 DO
 170 FOR F=1 TO 5
 180 INPUT (V$(F)),N$(F)
 190 EXIT IF N$(F, TO 1)=" STOP
"
 200 PRINT #4;N$(F)
 210 NEXT F
 220 LOOP : CLOSE #4: END PROC
 230 DEF PROC IN: CLS
 240 OPEN #4;"M";1;"BESTAND"IN
 260 DO
 270 FOR F=1 TO 5: PRINT #4;
 280 EXIT IF USR 432=0
 290 INPUT #4;N$(F)
 300 PRINT AT F*2+1,1;N$(F)
 310 NEXT F: PAUSE 100
 320 LOOP : CLOSE #4: END PROC
 330 DEF PROC EXP
 340 OPEN #4;"M";1;"BESTAND"EXP
 350 PROC UIT
 360 END PROC
```

CONVERSIE VAN DECIMAAL NAAR HEXADECIMAAL.

Deze funktie zet decimale getallen om tot hexadecimalen. Getallen tussen -255 en 255 resulteren in een hexadecimaal met twee digits, deze groter dan 255 hebben een LEN 4.

Getallen boven 65535 gaven 11, 'B Integer out of range'.

```
FN H$(32)    = "20"
FN H$(255)   = "FF"
FN H$(-64)   = "C0"
FN H$(-1024)= "FC00"
```

Bij voorbeeld

```
  10 RESTORE
  20 FOR F=1 TO 8: READ DATA
  30 PRINT AT 10,1;" BU ";F;" ";
 USING "00000";DATA;" DEC= ";HEX
$(DATA);" HEX"
  40 PAUSE 200
  50 NEXT F
  60 DATA 0,1,10,11,16,255,256,6
5535
```


INSTRING maakt het mogelijk om zeer snel string a te doorzoeken op gelijkenis met string b. Beide strings mogen gedimensioneerd zijn.
Het startadres geeft de positie(s) aan van string a in string b
Het volgende voorbeeld laat u de werking zien:

```
  10 LET A$="ABCDEFGHIJKLMNOPQRS
TUVWXYZ"
  20 LET A$=STRING$(10,A$)
  30 LET B$="E"
  40 LET TEST=1
  50 DO
  60 LET TEST=INSTRING(TEST,A$,B
$)
  70 EXIT IF TEST=0
  80 PRINT " A$ ";"(";TEST;")",
  90 LET TEST=TEST+1
 100 LOOP
 110 STOP
```

A$ bevat bijna het hele alfabet en wordt vervolgens 10 maal gekopieerd. B$ is de te vinden string en bevat de letter E. Variabele TEST krijgt de waarde 1. In de DO LOOP lus wordt TEST steeds met 1 verhoogd. Als B$ in A$ wordt gevonden, krijgt TEST de waarde van de betreffende positie in A$. Wordt B$ niet (meer) gevonden dan krijgt TEST de waarde 0 en EXIT IF zorgt er dan voor dat de lus verlaten wordt. Regel 80 leert u zien op welke posities de B$ in A$ voorkomt.

De LEN van A$ is (voor zover het geheugen van uw computer het toestaat) onbeperkt, terwijl de LEN van B$ niet meer dan 256 mag zijn, anders krijgt U foutmelding 10, A 'Invalid argument'.

Variabele TEST in het programma kan de waarde 0 ook verkrijgen wanneer B$ langer dan A$ zou zijn of wanneer het opgegeven startadres hoger is als de LEN van A$, of wanneer een van beide strings LEN 0 heeft.

Sommige karakters van B$ mogen vervangen worden door het " # " teken, hetgeen dan alle mogelijke karakters voorstelt, tenzij dit teken het allereerste teken in B$ is. Pas het programma als volgt aan:

```
30 LET B$ = " B#D " :
80 PRINT "A$ "; "("; TEST;  " TO "; TEST + LEN B$ - 1;
   ")",
```

Nu zal "BAD", "BBD" enzovoort gevonden worden indien aanwezig. N.B. De printroutine vertraagt maar laat u uiteindelijk zien waar het om gaat.

```
30 LET B$ = "#BD "
```

zal INSTRING laten zoeken naar het " # " teken en niets vinden.

POKE (strings) , MEMORY$ () en INSTRING vormen samen een zeer sterk trio. Voorbeelden van gecombineerd gebruik vindt u bij de voorbeeld-programma's.

Beschrijving van het voorbeeldprogramma.

Het volgende programma toont U hoe een string gecreeerd wordt en weggePOKEd wordt. Vervolgens worden de variabelen geCLEARd, zodat de string niet langer meer bestaat. Vervolgens wordt de string opnieuw opgebouwd (regel 440-470) en dient een deeltje van de string teruggevonden te worden (regel 540-570).

DPOKE 39998, LEN A$ op regel 240 doet het volgende: (stel de LEN a$ op 700 bijvoorbeeld)

```
POKE 39998, 700-  INT (700/256)*256        = 188
POKE 39999, INT   (700/256)                =   2

PEEK 39998 geeft 188 en
PEEK 39999 geeft 2.

DPEEK 39998 = 700 !  en  188 + (256*2)= 700
```

DPOKE zorgt er dus in dit geval voor dat de LEN van e$ op twee adressen wordt weggePOKEt, zonder dat U een en ander moet berekenen.

POKE 40000, A$ op regel 280 sleept de gehele A$ op vanaf adres 40000 tot en met 40000 + LEN A$.

Regel 540 geeft variabele B het adres waar A$ (7 TO 10) weggePOKEd werd.

VOORBEELDPROGRAMMA: INSTRING

```
  10 REM ** geheugen reserveren **
  20 CLEAR 39997
  50 REM ** lange string maken **
  60 LET A$="  EEN ZEER LANGE ST
RING OM WEG TE POKEN  "
  70 LET A$=A$+A$+A$+A$+A$
  80 LET A$=A$+A$+A$
 110 REM ****PRINTEN A$******
 120 PRINT A$
 130 PAUSE 200
 140 CLS
 170 REM ** LEN A$ PRINTEN *****
 180 PRINT "LEN A$ IS ";LEN A$
 190 PAUSE 200
 200 CLS
 230 REM LENGTE A$ WEGPOKEN
 240 DPOKE 39998,LEN A$
 270 REM A$ WEGPOKEN
 280 POKE 40000,A$
 310 REM ALLE VARIABELEN CLEAREN
 320 CLEAR
 350 REM A$ BENOEMEN
 360 LET A$=""
 390 REM VARIABELE LEN WAARDE GE
VEN VAN WEGGEPOOKTE LEN A$
 400 LEN=DPEEK(39998)
 430 REM HET WEGGEPOOKTE TERUGHA
LEN EN A$ VERLENGEN
 440 FOR F=1 TO LEN
 450 LET A$=A$+CHR$ PEEK (39999+
F)
 460 PRINT CHR$ PEEK (39999+F);
 470 NEXT F
 480 PAUSE 200
 490 CLS
 520 REM HET WOORD "ZEER" TERUGV
INDEN
 530 REM "ZEER"  IS A$(7 TO 10)
 540 LET B=INSTRING(40000,MEMORY
$(),A$(7 TO 10))
 550 FOR A=B TO B+(LEN A$(7 TO 1
0))
 560 PRINT CHR$ PEEK A;
 570 NEXT A
```

MEM() geeft aan hoeveel bytes er nog vrij zijn. Dit is een heel eenvoudige funktie daar zij hoofdzakelijk bestaat uit een CALL naar een ROMroutine. Het equivalent in Basic kan dit illustreren. Er is geen geschikte systeemvariabele voorhanden.

PRINT 65535 - USR 7962 wordt in Beta-Basic : PRINT MEM()

Beide instrukties na elkaar ingebracht als directe opdrachten geven steeds een klein verschil. Dit is te wijten aan het verschil in lengte van de opdrachten zelf.
Het is nuttig te beschikken over gegevens aangaande het gebruik van het geheugen. We kunnen het aantal bytes dat door Basic wordt gebruikt als volgt berekenen:

DPEEK (23627) - DPEEK (23635)

We hebben dit reeds uitvoerig besproken in hoofdstuk 3.9 DPOKE.

Beschrijving Informatie programma

Met deze informatie-routine kunt tijdens het programmeren steeds beschikken over allerlei handige informatie. Het is de bedoeling het steeds VOORAF te MERGen en het CLOCK-statement in Uw programma op te nemen in plaats van in deze routine.

CLOCK "" zet de klok van de Spectrum op 00:00:00 bij het passeren van deze instructie. Het is dus noodzakelijk dat deze instructie eenmalig ingelezen wordt (wellicht DELETEn !)
Deze routine kan uitstekend als PROC(edure) gedefineerd worden en onder een DEF KEY ingebracht worden.

```
DEF KEY "i" : PROC info
```

Telkens U SYMBOL SHIFT + BREAK/SPACE en dan i geeft zal de PROC(edure) aangeroepen worden,
Het programma bevat overigens voorbeelden van het gebruik van DPEEK en de systeemvariabelen, USING en de TIME$() functie.
Door middel van RENUM LINE 9990 STEP 1 kan deze routine ook achteraan het programma gezet worden.

```
  10 CLOCK "": CLS : PRINT "PROG
RAMMA INFORMATIE"
  40 PRINT AT 4,0;"TOTAAL RAM";T
AB 20; USING "#####";(DPEEK(2373
2)-16383)/1024;" Kb "
  50 PRINT AT 6,0;"MICRODRIVE MA
PS";TAB 20; USING "#####";DPEEK(
23631)-23734;" BYTES"
  60 PRINT AT 8,0;"BASIC PROGRAM
MA";TAB 20; USING "#####";DPEEK(
23627)-DPEEK(23635);" BYTES"
  70 PRINT AT 10,0;"VARIABELEN";
TAB 20; USING "#####";DPEEK(2364
1)-DPEEK(23627)-1;" BYTES"
  80 PRINT AT 12,0;"GEHEUGEN VRI
J";TAB 20; USING "#####";MEM();"
 BYTES"
  85 PRINT AT 14,0;"RAMTOP";TAB
20; USING "#####";DPEEK(23730);"
 ADRES"
  90 LET I$=TIME$(): PRINT AT 16
,0;"TIJD AAN";TAB 17;I$;" UUR"
```

Beschrijving van voorbeeldprogramma 2

Het informatie-programma van de vorige bladzijde kan in zijn geheel ook ingebracht worden als DEFKEY. U dient daartoe regel 10 te EDITten en met SPLIT (<>) de instructie CLOCK "" op regel 5 te zetten. Vervolgens EDIT u regel 10 opnieuw en voegt u als eerste instructie toe: DEF KEY "1" :

Nadien dient U alle regels na regel 10 aan regel 10 te JOINen, zodat uiteindelijk slechts twee regels onstaan. Om de DEF KEY iets in te korten kunt u desgewenst de meeste : PRINT AT instructies vervangen met ; AT.

Nu kunt U regel 20 toevoegen an het programma plus de (nieuwe) BB code SAVEn, op de manier zoals is voorzien in de regels 1 en 2 als U Beta-Basic inlaadt. Deze twee regels worden altijd geDELETEd; om dit te voorkomen dient U dus alvorens dit programma in te tikken eerst MERGE "Beta-Basic" te geven en na het MERGEn in regel 2 de instructie DELETE 1 TO 2 te verwijderen. Laad nu de BB code in en tik het programma in.

Laat nu het programma RUNnen.
Als U nu SYMBOL SHIFT + BREAK/SPACE en 1 geeft wordt het programmaatje uitgevoerd. U kunt nu een eigen uitgebreide versie van de BB code SAVEn en later in uw andere programma's gebruiken. U dient het programma(tje) en BB code dan te SAVEn door GOTO 1 te geven.

Om in Uw eigen programma deze uitgebreide BB te laden handelt u als volgt :

LOAD "" en Uw eigen Beta-Basic versie wordt ingeladen.

Nu kunt u Beta-Basic gebruiken zoals voorheen met als extra de informatie onder DEF KEY 1.

PROGRAMMA:

```
   5 CLOCK ""
  10 DEF KEY "1": CLS : PRINT "P
ROGRAMMA INFORMATIE";AT 4,0;"TOT
AAL RAM";TAB 20; USING "######";(
DPEEK(23732)-16383)/1024;" KB";A
T 6,0;"MICRODRIVE MAPS";TAB 20;
USING "######";DPEEK(23631)-23734
;" BYTES";AT 8,0;"BASIC PROGRAMM
A";TAB 20; USING "######";DPEEK(2
3627)-DPEEK(23635);" BYTES";AT 1
0,0;"VARIABELEN";TAB 20; USING "
######";DPEEK(23641)-DPEEK(23627)
-1;" BYTES";AT 12,0;"GEHEUGEN VR
IJ";TAB 20; USING "######";MEM();
" BYTES";AT 14,0;"RAMTOP";TAB 20
; USING "######";DPEEK(23730);" A
DRES": LET I$=TIME$(): PRINT AT
16,0;"TIJD AAN";TAB 17;I$;" UUR"
  20 DELETE 10 TO 20
```

MEMORY $ () is een zeer handige functie die het geheugen omzet in een string. De LEN van MEMORY$ is 65532, de eerste byte en de laatste drie zijn om technische redenen niet opgenomen.

In combinatie met het POKEn van strings en INSTRING is het mogelijk om zeer snel grote delen van het geheugen te verplaatsen.
De volgend voorbeelden tonen aan waarvoor MEMORY$() onder andere te gebruiken is:

```
  10 LET A$=" BETA BASIC "
  20 LET A$=STRING$(5,A$)
  30 LET ADRES=40000
  40 POKE ADRES,A$
  50 LET B$=A$(7 TO 9): REM"BAS"
  60 DO
  70 LET TEST=INSTRING(ADRES,MEM
ORY$(),B$)
  80 EXIT IF TEST=0
  90 PRINT MEMORY$()(TEST TO TES
T+LEN B$-1)
 100 LET ADRES=TEST+LEN B$
 110 LOOP
 120 STOP
```

A$ wordt op adres 40000 weggePOKEt en een deel van A$ (B$) moet worden opgezocht in het geheugen. Er wordt gezocht in het gebied vanaf 40000 (regel 70); in plaats van adres 40000 kan ook 1 opgegeven worden, zijnde het begin van het geheugen of DPEEK (23635), zijnde de PROG systeemvariabele.
INSTRING is zo snel dat het niet erg veel uitmaakt vanaf welk adres u begint te zoeken. Om dit aan te tonen dient het programma als volgt aangepast/uitgebreid te worden:

```
  55 LET ADRES=1: LET AANTAL=0
  75 LET AANTAL=AANTAL+1
  90 PRINT MEMORY$()(TEST TO TES
T+LEN B$-1);" ";TEST;"  TOT "; T
EST+LEN B$-1;" AANTAL ";AANTAL
```

RUN het programma en alle adressen waar B$ voorkomt worden u getoond.

Variabele TEST krijgt de waarde van de adressen waar B$ teruggevonden is. Als B$ niet (meer) gevonden wordt krijgt TEST de waarde 0 en wordt er uit de lus gesprongen.

```
  10 REM BETA BASIC
  20 PRINT INSTRING (1,MEMORY$(),
"BETA BASIC")
OF:
  10 REM BETA BASIC
  20 PRINT INSTRING(DPEEK(23635)
,MEMORY$(),"BETA BASIC")
```

Zet de gehouganplaatsen gevan van "BETA BASIC" in het REM statement.
MEMORY$() wordt ook gebruikt om de DISPLAY en ATTRIBUTEN file op te slaan in strings en om een heel programma weg te POKEn. Zie ook de voorbeelden die bij POKE zijn beschreven en de voorbeeld-programma's.

4.11 MOD (getal,getal) FN UC

MOD geeft het restgetal aan welke overblijft wanneer het eerste getal wordt gedeeld door het tweede. Men noemt dit het eerste getal modulo het tweede nemen. In Basic is het equivalent daarvan:

```
 10 REM SPECTRUM BASIC VERSIE
 20 INPUT "GETAL 1 ";GETAL1
 30 INPUT "GETAL 2 ";GETAL2
 40 PRINT GETAL1;" / ";GETAL2;"
HEEFT ALS REST ";GETAL2*(GETAL1
/GETAL2-INT (GETAL1GETAL2))

 50 REM IN BETA BASIC
 60 PRINT MOD(GETAL1,GETAL2)
```

We kunnen MOD handig binnen een lus gebruiken en bv. zeggen : indien de rest van a/b = 0 THEN ...

```
 80 FOR F=1 TO 50
 90 IF MOD(F,2)=0 THEN PRINT F;
" TWEEVOUD"
100 IF MOD(F,10)=0 THEN PRINT F
;" TIENVOUD"
```

4.12 NUMBER (2-karakterstring) FN NC

Deze funktie zet een twee-karakterstring om tot een integer getal tussen 0 en 65535. Het Basic-equivalent is:

```
LET getal = 256* CODE C$(1) + CODE C$(2)
```

Indien de stringlengte LEN anders is dan 2, dan krijgt u foutboodschap 10, 'A Invalid argument'.

Deze funktie wordt aangewend om de 2-karakterstrings welke ontstonden na conversie met de funktie CHR$, opnieuw tot getallen om te zetten. Ook bij CHR$ kunt u een voorbeeld vinden.

VOORBEELDPROGRAMMA:

```
 10 REM SPECTRUM BASIC
 20 LET C$=" SCROLL DO "
 30 LET GETAL=256*CODE C$(1)+CO
DE C$(2)
 40 PRINT GETAL

 50 REM BETA BASIC
 60 LET C$=" DO SCROLL "
 70 PRINT NUMBER(C$)
```

4.13 OR (getal, getal) FN O(

Net als de funktie AND verschilt ook de Beta-Basic OR van deze van de Spectrum door andere syntax. OR(getal, getal) vergelijkt de twee getallen 'bit-na-bit', waarbij het bitresultaat 1 wordt, indien een van de bits 1 is. Is in beide getallen eenzelfde bit gelijk aan 0, dan is het bitresultaat daarvan 0. De getellen dienen te liggen tussen 0 en 65535.

```
      OR (80,250)    d    h

      01010000      80   50
      11111010     250   FA
   =  11111010     250   FA
```

Dit houdt o.m. in dat elke OR met het getal 0 resulteert in het andere getal; dat OR met 65535 altijd zal resulteren in 65535 en dat een OR met 255 en het tweede getal kleiner dan 255, als resultaat 255 heeft.

```
      OR (45,255)                    OR (0, 137)

      00101101      45    2D         00000000      0   00
      11111111     255    FF         10001001    137   89
   =  11111111     255    FF      =  10001001    137   89
```

VOORBEELDPROGRAMMA:

```
10 LET A=180: LET B=250
20 LET GETAL=OR(A,B)
30 PRINT BIN$(A);" ";HEX$(A);" ";A
40 PRINT BIN$(B);" ";HEX$(B);" ";B
50 PRINT BIN$(GETAL);" ";HEX$(GETAL);" ";GETAL
```

4.14 RNDM (getal) FN R(

Deze funktie is in feite precies hetzelfde als RND. De syntax is evenwel verschillend en RNDM werkt ongeveer 2.5 maal sneller dan RND en RNDM geeft steeds integere getallen, met uitzondering van RNDM (0).

```
PRINT RNDM (0)       geeft een willekeurig niet interger getal
                     tussen 1 en 0
PRINT RNDM (1)       geeft 1 ofwel 0 als resultaat
PRINT RNDM (-10)     geeft een willekeurig getel tussen -10 en 0
```

Het getal achter RNDM behoort zelf ook tot de mogelijke resultaten, terwijl dit bij RND niet zo is.
RANDOMIZE (getal) zet RNDM (net als RND) op een vaste pleats waar het "trekken" van de willekeurige getallen moet worden begonnen.

VOORBEELDPROGRAMMA:

```
10 DO
20 PRINT AT 0,0;"SPECTRUM RND ";INT (RND*6)+1''
30 PRINT AT 2,0;"BETA BASIC RNDM ";RNDM(6)'
40 EXIT IF INKEY$="" STOP "
50 PAUSE 20: LOOP
70 STOP
```

4.15 SCRN$ (regelnummer, kolom) FN K$(

SCRN$ is in feite precies hetzelfde als SCREEN$ van de Spectrum. het gaat na wat er op het scherm staat. De Spectrum SCREEN$ herkent echter geen zelfgedefinieerde GRAPHICS, terwijl SCRN$ dit wel doet.

Het volgende programma toont dit aan:

```
 10 KEYWORDS 0
 20 DATA 0,1,2,60,84,84,20,20
 30 FOR N=0 TO 7
 40 READ DATA
 50 POKE USR "S"+N,DATA
 60 NEXT N
 70 PRINT AT 10,0;STRING$(32,"
SCROLL "): REM KEYWORD SCROLL
 80 LET A$=""
 90 FOR N=0 TO 31
100 LET A$=A$+SCRN$(10,N)
110 NEXT N
120 PRINT A$
130 KEYWORDS 1
140 FOR N=1 TO 300
150 ROLL 8
160 NEXT N
```

RUN het programma en vervang nadien op regel 100 SCRN$ door SCREEN$ en merk het verschil op.

SCRN$ herkent echter de GRAPHICS van de Spectrum niet. Indien u deze nodig heeft, dient u ze zelf te definieren.

4.16 SINE (getal) FN S(

Geeft de sinus van het aangegeven getal zoals SIN uit Spectrum-Basic , maar net als bij COSE met een geringere nauwkeurigheid van 4 cijfers na de komma en de berekening is ongeveer 60% sneller.

VOORBEELDPROGRAMMA:

```
 10 CLOCK 1: CLOCK ""
 20 FOR F=0 TO 255
 30 PLOT F,88+80*SINE (F/128*PI)
 40 NEXT F
 50 PRINT AT 21,0; TIME$()
 60 STOP
```

Laat dit programma lopen met SINE van BETA BASIC en noteer de tijd, vervang nu SINE door SIN van het SPECTRUM BASIC en vergelijk de tijdan.
ZIE OOK: 4.4 COSE

4.17 STRING$ (n, string) — FN S$(

De funktie FN S$ PRINT de opgegeven string een n aantal maal na elkaar. Zo kan je snel een lijntje trekken met :

```
PRINT STRING$ (32,"-")
```

Dat is ook nuttig te gebruiken om een of meerdere lijnen op het scherm te wissen. PRINT AT 20,0;STRINGS (64," ") zal lijnen 20 en 21 met spaties overschrijven.
Indien in bovenstaande lijn een tijdelijke PAPERkleur wordt aangegeven, dan kan je ook met STRINGS snel een deel van het scherm inkleuren.

```
PRINT PAPER 4, INK 6, BRIGHT 1; STRINGS (704,"*")
```

Zet het gehele scherm vol sterretjes.

```
LET A$="  ---  ": PRINT STRING$ (5,A$)
```

We kunnen een string verlengen met:

```
10 LET A$="Z"
20 LET A$=STRING$ (10,A$)
30 PRINT A$
```

4.18 TIME$() — FN T$(

Deze funktie geeft de huidige tijd aan van de Beta-Basic CLOCK. Het is zeer handig om in het begin van uw programma bv. CLOCK "" op te nemen en bij het beeindigen van het programma (of deel) de tijd op te vragen.
Bij het testen van programmadelen of procedure's kunnen we de tijdsverschillen vastleggen, zoals in onderstaand voorbeeld. Op analoge wijze kan u het snelheidsverschil nagaan tussen : RNDM en RND; SINE en SIN; en COSE en COS.

VOORBEELD 1:

```
10 CLOCK ""
20 FOR F=1 TO 500: NEXT F
30 LET A$=TIME$()
40 CLOCK ""
50 LET F=1
60 DO UNTIL F=500
70 LET F=F+1
80 LOOP
90 B$=TIME$()
100 PRINT "FOR NEXT LUS ";A$
110 PRINT "DO LOOP LUS  ";B$
120 PRINT "VERSCHIL IS ";VAL B$(7 TO 8)-VAL A$(7
 TO 8);" SECONDEN":STOP
```

VOORBEELD 2:

```
10 CLOCK "": CLOCK 1
20 DO
30 LET T$=TIME$()
40 PRINT AT 1,21;"UREN      ";T$( TO 2);
50 PRINT TAB 21;"MINUTEN  ";T$(4 TO 5);
60 PRINT TAB 21;"SECONDEN ";T$(7 TO 8)
70 EXIT IF INKEY$=" STOP " OR T$="00.00.20"
80 LOOP
90 PRINT AT 16,0;"PROGRAMMA ONDERBROKEN IE "'T$;
" UUR"
```

4.19 XOR (getal, getal) FN XO

Deze funktie geeft een zogenaamde exclusieve OR van de beide getallen. Deze dienen eveneens te liggen tussen 0 en 65535. De 'bit-na-bit' vergelijking resulteert in 0 indien eenzelfde bit in beide getallen 1 of 0 is . Is een bit 1 in een van beide getallen, dan wordt het resultaat 1.

XOR (168,67)	dec	hex	XOR (207,195)	dec	hex
10101000	168	A8	11001111	207	CF
01000011	67	43	11000011	195	C3
= 11101011	235	EB	= 00001100	12	0C

Het valt op dat 235-67=168 en dat 207-195=12 maar let op de volgorde der getallen! Maar dit is niet altijd het geval.

XOR (156,68)	dec	hex
100111000	156	9C
010001000	68	44
= 110110000	216	D8

PROGRAMMAATJE:

```
10 PRINT BIN$(XOR(BIN 11111111,BIN 11110000))
20 PRINT BIN$(XOR(BIN 00001111,BIN 11110000))
30 PRINT BIN$(XOR(BIN 11100111,BIN 11110010))
```

5. VARIABELEN.

A. xos, xrg, yos, yrg.

De variabelen van Beta-Basic dienen per letter ingetoetst in grote of kleine letters.
Xos en yos vormen de startpositie in het 0 tot 255 op 0 tot 175 pixelgroot (x,y)-assenstelsel voor PLOT-, DRAW-, en CIRCLE- instrukties. Bij aanvang staan xos en yos op (0,0), waarbij gePLOT wordt tot (255,175). Dit komt overeen met de Spectrum aanvangs-startpositie bij 'PLOT-instrukties'. De systeemvariabelen COORD-x en COORD-y onthouden het laatst gePLOTte beeldpunt en hun aanvangswaarde is (0,0).
Maken we xos en yos groter dan (0,0), bij voorbeeld (128,88). Er wordt nu gePLOT vanuit het midden van het scherm (zie de figuur van GRAPHIC 6) en wel van 128 tot -127 op de x-as en van 88 tot -87 op de y-as. Het middelpunt van het scherm is nu punt (0,0).

(xos,yos) = (startpositie op x-as, startpositie op y-as)

Met xrg en yrg wijzigen we het kader van de 'PLOT-instrukties'. Dit kader staat in standaardomstandigheden op 255 voor xrg en 175 voor yrg. Halveren we de kadergrootte tot xrg 128 en yrg tot 88. Er kan nu maximaal 'gePLOT' worden tot 127 op de x-as en 87 op de y-as.

xrg = maximaal rechtse 'PLOT-kantlijn' of raambreedte
yrg = hoogste y-as kantlijn of raamhoogte

Bij het verkleinen van het raam of de kader, gebeurt dit naar de linker benedenhoek toe . Dit is te zien in het voorbeeld: het hoekpunt van een rechthoek verplaatst zich diagonaal.

```
10 FOR F=255 TO 30 STEP -14.02
20 LET XRG=F: LET YRG=F-.5
40 PLOT 0,0
50 DRAW 30,0: DRAW 0,30
60 DRAW -30,0: DRAW 0,-30
70 NEXT F
```

Bij willekeurige wijziging van het plotraam zullen de verhoudingen van een 'geplotte' figuur veranderen! Dank eraan dat figuren getekend in een groot kader en/of met laag startpunt al snel te groot worden bij wijziging (verkleinen) van het kader en bij het verleggen van de assen.

De Beta-Basic variabelen worden bij een CLEAR of RUN opdracht niet gewist, doch krijgen dan specifieke waarden toegekend. CLEAR een keer en geef dan PRINT xos en u krijgt "0". Niet de te verwachten 2, '2 Variable not found'. Het startpunt krijgt na RUN of CLEAR de coordinaten (0,0). Dit wil zeggen: het startpunt is hetzelfde als in Spectrum-Basic.
Het nut van xos en yos, of van het verleggen van het startpunt op het assenstelsel, is dat men o.a. sneller een startpunt kan wijzigen dan een hele tekening, figuren in verhoudingen kan laten wisselen, of geplotte tekst zou kunnen laten bewegen?!

B. error, line, stat en angle.

De variabele error geeft de waarde van de foutboodschappen aan en moet steeds gebruikt worden in samenhang met het ON ERROR statement. We verwijzen naar het hoofdstukje 3.19 voor verdere verklaringen. De waarden van de foutmeldingen vind u terug in appendix 5.
De funktie van de variabele angle is ons helaas niet met zekerheid bekend.

6.1 Programma tijden-teller

Procedure tijden kan van nut zijn voor het samentellen van uren en minuten. Het toont de werking van de functie MOD en het gebruik van USING.

Door de FOR-NEXT lus (regel 80) te verhogen kunt u het aantal in te brengen tijden verhogen.

TIJDENTELLER

```
    1>LET A$="TIJD"
    2 ERASE "M";1;A$
    3 SAVE *"M";1;A$
    4 VERIFY *"M";1;A$
    5 STOP
   10 DEF PROC tijden
   20 CLS
   30 PRINT AT 1,14;"0:00"
   40 LET uur=60
   50 LET toturen=0
   60 LET totminuten=0
   70 LET regel=1
   80 FOR x=1 TO 30
   90 PRINT INVERSE 1; BRIGHT 1;
INK 1;AT 20,0;"  VOER  s    IN OM
 TE STOPPEN     "
  100 PRINT INVERSE 1; BRIGHT 1;
INK 2;AT 21,0;"  VOER  *    IN OM
 TE HERSTARTEN "
  110 INPUT "aantal uren "; LINE
a$: IF a$="s" THEN STOP
  120 IF a$="*" THEN GO TO 20
  130>IF CODE a$<42 OR CODE a$>58
 THEN PRINT #0;AT 1,17; BRIGHT 1
; FLASH 1;" ENKEL CIJFERS ": PAU
SE 0: GO TO 110
  140 PRINT AT 21,0;STRING$(32,"
")
  150 PRINT AT 20,0;STRING$(32,"
"): INPUT "aantal minuten "; LIN
E f$
  160 IF CODE f$<42 OR CODE f$>58
 THEN PRINT #0;AT 1,17; BRIGHT 1
; FLASH 1;" ENKEL CIJFERS ": PAU
SE 0: GO TO 150
  170 LET uren2=VAL a$: REM uit i
nput
  180 LET minuten2=VAL f$: REM id
em
  190 PRINT AT regel,0; USING "00
00";uren2;":";TAB 5; USING "00";
minuten2
  200 LET toturen=toturen+uren2
  210>LET totminuten=totminuten+m
inuten2
  220 LET totaal=totminuten/uur
  225 LET totaal=INT totaal
  230 LET tot2=MOD (totminuten,uur
)
  240 LET totaal=toturen+totaal
  250 PRINT OVER 1;AT regel,11; U
SING "0000";totaal;":";TAB 16; U
SING "00";tot2
  260 LET regel=regel+1: IF regel
=19 THEN BEEP 1.1: PAUSE 0: CLS
: LET regel=1
  270 NEXT x
  280 END PROC
```

6.2 Programma DEF KEY overzicht

Onderstaand programma toont U het gebied van een DEF KEY en laat U de inhoud en de CHR$'s per adres zien.

Er werd gebruik gemaakt van het PLOTten van strings om de onderlinge letterafstand te vergroten.

Variabele rt geeft het adres aan waar het gebied van de DEF KEY begint. Het begin bestaat altijd uit een positie voor beginmerkering (groter dan teken, >), een positie voor de toets waaronder de DEF KEY werd ingebracht, twee posities voor de LEN van de DEF KEY en vervolgens de instructie of de string zelf.

PRINT DPEEK (23730)+2 geeft de LEN van de laatst ingebrachte DEF KEY. Dit adres mag niet gePOKEd worden aangezien de LEN informatie hierin opgeslagen is.

POKE (23730) + 1, 2 (of andere waarde) heeft tot gevolg dat de toets die ingedrukt moet worden, gewijzigd wordt.

DEF KEY

```
  20 CLEAR 55400
  30 LET a$="BETA": DEF KEY "2",
a$
  40 LET rt=DPEEK(23730): LET x=
55400
  50 PROC uitlezen: STOP
  60 DEF PROC uitlezen
  70 LET y=15  LET z=3. PROC kad
er
  80 FOR a=rt TO x
  90 LET a$=CHR$ PEEK a
 100 IF CODE a$<32 THEN LET a$="
?"
 110 PROC plot: LET z=z+25
 120 NEXT a
 130 LET y=40: LET z=3
 140 FOR a=rt TO x
 150 LET astr=PEEK a
 160 LET a$=STR$ astr
 170 PROC plot: LET z=z+24
 180 NEXT a: LET pos=1
 190 FOR a=rt TO x: LET reg=9
 200>LET a$=STR$ a
 210 FOR g=1 TO LEN a$
 220 PRINT AT reg,pos;A$(G)
 230 LET reg=reg+1: NEXT g
 240 LET pos=pos+3: NEXT a
 250 PRINT AT 15,27;"BB"
 260 END PROC
 270 DEF PROC plot
 280 PLOT OVER 1,z,y,a$
 290 END PROC
 300 DEF PROC kader
 310 PLOT 0,0: DRAW 255,0
 320 PLOT 0,110  DRAW 255,0
 330 PLOT 0,0: DRAW 0,110
 340 FOR g=0 TO 200 STEP 24
 350 PLOT g,0: DRAW 0,110
 360 NEXT g
 370 PLOT 0,30  DRAW 192,0
 380 PLOT 0,60: DRAW 192,0
 390 PLOT 255,0  FOR g=0 TO 110
STEP 5
 400 PLOT 255,g
 410>NEXT g
 420 END PROC
```

6.3 Programma I TJING

Algemene beschrijving

Dit programma laat U orekalen met de Spectrum toe. De I Tjing (het Boek der Veranderingen) is van oorsprong Chinees en dateert van zolang China bestaat.

Oorspronkelijk werd er gerakeld met duizendbladstelen of met munten. Deze technieken zijn in het programma software-matig vervangen door tijdens het doorlopen van een korte FOR-NEXT lus te ENTERen, waardoor steeds willekeurig een "worp" wordt gedaan.

Na het werpen wordt er aangegeven welke worp men gedaan heeft en of de worp een "sterke" yin of yang is. Na de zesde worp zoekt de Spectrum uit het intern geheugen het juiste hexagram op en geeft U het nummer van het hexegram op, alsmede de chinese en nederlandse titel. Aan de hand hiervan kunt U direct in de I Tjing boeken Uw worp naslaan.

De boeken door de auteurs geraadpleegd zijn:

ORAKELEN MET MUNTEN	van Da Liu	ISBN 90 6019373 3
I TJING	van R.Wilhelm	ISBN 90 2024785 9
SLEUTEL TOT DE I TJING	van H.van Praag	ISBN 90 2024790 5

BESCHRIJVING VAN HET GEBRUIK

De DATA worden NIET in Beta-Basic ingevoerd maar in gewoon Spectrum-Basic. Dit omdat er grafische DATA ingevoerd dienen te worden. Tik listing 2 in en SAVE de DATA zoals op regel 7000 of 7010 staat aangegeven.

De DATA bestaan uit zes reeksen van zes blokjes (GRAPHICS 8 en spaties) gescheidan door een # teken en vervolgens de titels.

Voor de zekerheid kunt U eveneens het programma van listing 2 SAVEn en VERIFYen op de gebruikelijke manier.

Laad nu de Beta-Basic en de BB code. Aangezien dit programma gebruik maakt van GRAPHICS is het nodig dat U regelmatig van KEYWORDS wisselt.

Tik regel 5 tot en met 220 in. Vervolgens zet U de Spectrum in de KEYWORDS 0 mode en tikt U regel 230 tot en met 310 in. Nu zet U de Spectrum terug in KEYWORDS 1 en tik regel 320 tot en met 520 in. Wederom zet U de Spectrum terug in KEYWORDS 0 en tikt U regel 530 tot 800 in. De regels 811 tot en met 930 moeten weer in KEYWORDS 1 ingetikt worden.

Laad nu de DATA in met de opdracht:

LOAD "" DATA h$() voor cassette en
LOAD *"M";1," DATA itjing" DATA h$() voor drivas

Geef nu GOTO 860 en het geheel (hoofdprogramma, BB en de DATA) worden achter elkaar geSAVEd. Denk eraan om een nieuwe cassette of cartridge te gebruiken; de eerder gebruikte cassette of cartridge kan dan gewist of geERASEd worden.

Nu kan het programma opgeladen worden door:

LOAD "" voor cassette en
LOAD *"M";1;"I Tjing" voor drives

Na het LOADen wordt automatisch de BB code en de DATA geLOADt en start het programma.

DEF PROC overzicht geeft een overzicht van alle hexegrammen. Deze kan aangeroepen worden door:

PROC overzicht <ENTER> in te tikken.

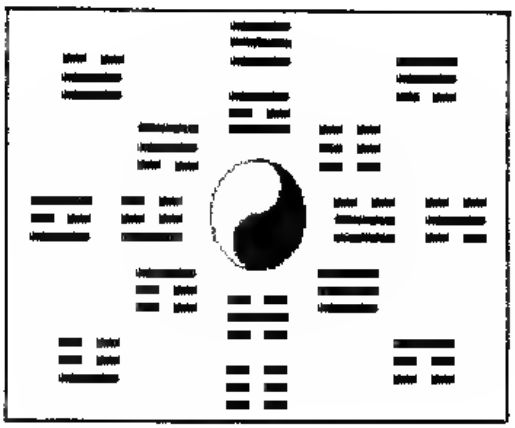
De gebruikte Beta-Basic bestaat in dit programma uit het KEYWORDS-statement wat ervoor moet zorgen dat de GRAPHICs afgedrukt worden. De afgebeelde listing (1) krijgt U dus niet als zodanig op het scherm. De functie INSTRING op regel 360 zoekt in de DATA naar Uw worpen.

De FILL instructie op regel 811 zorgt ervoor dat het " tai-chi " teken gedeeltelijk geFILLd wordt.

Regel 815 dient om het SCREEN plaatje van de " tai-chi" op te slaan in het geheugen onder a$.

DELETE op regel 824 verwijdert alle regels van 530 tot en met 824 (inclusief zichzelf), zodat de procedure teichl enigszins verkort wordt.

Regel 825 is de enige regel in PROC(edure) taichl die overblijft. Deze POKEt het opgeslagen SCREEN pleetje van de " tai-chi" weer terug, zodat de procedure alsnog kan functioneren.



```
   5 KEYWORDS 0
  10 PROC begin
  20 PROC spelen
  40 DEF PROC begin
  50 CLS : PRINT AT 20,2;"© Serg
e Girard 1984/1985": PAUSE 30: C
LS
  60 PROC taichi: END PROC
  70 DEF PROC spelen
  80 PAUSE 0: CLS
  90 LET z=20: LET b=1: LET y=1
 100 DIM g$(1,42)
 110 PROC random
 120 DEF PROC random
 130 FOR d=0 TO 5
 140 PRINT BRIGHT 1;AT 5,0;"ENTE
R s om te stoppen ";AT 7,0;"ENTE
R voor een worp    "
 150 LET s$=INKEY$
 160 IF s$="s" THEN PROC taichi:
 PAUSE 0: STOP

 170 IF s$=CHR$ 13 THEN LET a=d:
 PROC lijnprint
 180 PRINT AT z,19;b;"e worp"
 190 NEXT d
 200 PROC random
 210 END PROC
 220 DEF PROC lijnprint

 230 IF a=0 THEN LET z$="▀▀▀▀▀▀▀"
: PRINT AT z,0;z$;" yang"
 240 IF a=1 THEN LET z$="▀▀   ▀▀"
: PRINT AT z,0;z$;" yin"
 250 IF a=2 THEN LET z$="▀▀   ▀▀"
: PRINT AT z,0;z$;" yin"
 260 IF a=3 THEN LET z$="▀▀▀▀▀▀▀"
: PRINT AT z,0;z$;" yang"
 270 IF a=4 THEN LET z$="▀▀▀▀▀▀▀"
: PRINT AT z,0;z$;" sterke yang"
 280 IF a=5 THEN LET z$="▀▀   ▀▀"
: PRINT AT z,0;z$;" sterke yin"
 290 LET g$(1,y TO y+6)=z$+"#"
 300 LET y=y+7
 310 LET z=z-2: LET b=b+1
```

```
 320>IF b>6 THEN PROC zoek
 330 END PROC
 340 DEF PROC zoek
 350 FOR f=1 TO 64
 360 IF INSTRING(1,h$(f, TO 42),
g$(1)) THEN PRINT AT 0,0;f;" ";h
$(f,43 TO ) : PAUSE 0: PROC spel
en
 370 NEXT f
 380 END PROC
 390
 400 DEF PROC overzicht
 410 FOR f=1 TO 64
 420 PRINT AT 14,13;h$(f, TO 6)
 430 PRINT AT 12,13;h$(f,8 TO 13
)
 440 PRINT AT 10,13;h$(f,15 TO 2
0)
 450 PPINT AT 8,13;h$(f,22 TO 27
)
 460 PRINT AT 6,13;h$(f,29 TO 34
)
 470>PPINT AT 4,13;h$(f,36 TO 41
)
 480 PRINT AT 0,0;f;" ";h$(f,43
TO )
 490 PAUSE 0
 500 NEXT f
 510 END PROC
 520 DEF PROC taichi

 530 PAUSE 1: CLS : PRINT AT 0,1
4;"▄▄▄▄▄▄"
 540 PRINT AT 1,14;"▄▄▄▄▄▄";AT 2,1
4;"▄▄▄▄▄▄"
 550 PRINT AT 2,3;"▄▄ ▄▄";AT 2,25
;"▄▄▄▄▄▄"
 560 PRINT AT 4,14,"▄▄▄▄▄▄";AT 3,3
;"▄▄▄▄▄▄"
 570 PRINT AT 3,25,"▄▄▄▄▄▄";AT 5,1
4;"▄▄ ▄▄"
 580 PRINT AT 4,3;"▄▄▄▄▄▄";AT 4,25
;"▄▄ ▄▄"
 590 PRINT AT 5,14;"▄▄▄▄▄▄";AT 6,8
;"▄▄▄▄▄▄"
 600 PRINT AT 6,14;"▄▄▄▄▄▄";AT 6,2
0;"▄▄ ▄▄"
 610 PRINT AT 7,8;"▄▄▄▄▄▄";AT 7,20
;"▄▄ ▄▄"
 620 PRINT AT 8,8;"▄▄ ▄▄";AT 8,20
;"▄▄ ▄▄"
 630 PRINT AT 10,1;"▄▄▄▄▄▄";AT 10,
7;"▄▄ ▄▄"
```

```
 640 PRINT AT 10,21;"▀▀ ▀▀";AT 10
,27;"▀▀ ▀▀"
 650 PRINT AT 11,1;"▀▀ ▀▀";AT 11,
7;"▀▀ ▀▀"
 660 PRINT AT 11,21;"▀▀▀▀▀▀";AT 11
,27;"▀▀▀▀▀▀"
 670 PRINT AT 12,1;"▀▀▀▀▀▀";AT 12,
7;"▀▀▀▀▀▀"
 680 PRINT AT 12,21;"▀▀▀▀▀▀";AT 12
,27;"▀▀ ▀▀"
 690 PRINT AT 14,8;"▀▀▀▀▀▀";AT 14,
20;"▀▀▀▀▀▀"
 700 PRINT AT 15,8;"▀▀ ▀▀";AT 15,
20;"▀▀▀▀▀▀"
 710 PRINT AT 15,14;"▀▀ ▀▀";AT 16
,8;"▀▀ ▀▀"
 720 PRINT AT 16,20;"▀▀▀▀▀▀";AT 16
,14;"▀▀▀▀▀▀"
 730 PRINT AT 17,14;"▀▀ ▀▀";AT 18
,3,"▀▀ ▀▀"
 740 PRINT AT 18,25;"▀▀▀▀▀▀";AT 19
,3,"▀▀ ▀▀"
 750 PRINT AT 19,25;"▀▀ ▀▀";AT 19
,14;"▀▀ ▀▀"
 760 PRINT AT 20,3;"▀▀▀▀▀▀";AT 20,
25;"▀▀ ▀▀"
 770 PRINT AT 20,14;"▀▀ ▀▀";AT 21
,14;"▀▀ ▀▀"
 780 CIRCLE 127,87.5,25
 790 PLOT 127.62.5
 795 DRAW 0,25,-PI
 800 DRAW 0,25,PI

 811)FILL 140,87
 815 LET a$=MEMORY$()(16384 TO 2
3295)
 820 PAUSE 200
 824 DELETE 530 TO 824
 825 POKE 16384,a$
 830 END PROC
 850 STOP
 860 SAVE *"m";1;" DATA itjing"
DATA h$()
 865 LET rt=DPEEK(23730). RANDOM
IZE USR 59904
 870 SAVE *"m",1;"I Tjing" LINE
900
 880 SAVE *"m";1;"BB"CODE rt+1,6
5367-rt  STOP
 900 CLEAR rt. LOAD *"m";1;"BB"C
ODE
 910 RANDOMIZE USR 58419
 920 LOAD *"m";1;" DATA itjing"
DATA h$()
 930 GO TO 5
```

```
    1 DIM h$(64,100)

    4 REM ****creatie van data***

    7 REM **********************
   10 LET a$="████████#████████#"
   15 REM **********************
   20 LET b$="███   ███#███   ███#"
   25 REM **********************
   30 LET c$="████████#███   ███#"
   35 REM **********************
   40 LET d$="███   ███#████████#"
 1000 REM *******data************
 1005
 1006
  110 DATA a$+a$+a$+"tj'ien het s
cheppende"
  120 DATA b$+b$+b$+"k'oen het on
tvangende"
  130 DATA c$+b$+c$+"tsjoen de aa
nvangsmoeilijkheid"
  140 DATA d$+b$+d$+"meng de jeug
ddwaasheid"
  150 DATA a$+c$+c$+"siu het wach
ten"
  160 DATA d$+d$+a$+"soeng de str
ijd"
  170 DATA d$+b$+b$+"sje het lege
r"
  180 DATA b$+b$+c$+"pi de aaneen
gesl otenheid"
  190 DATA a$+c$+a$+"siau tsj'oe
de temmende kracht van het Klein
e"
  200 DATA a$+d$+a$+"lu het optre
den"
  210 DATA a$+c$+b$+"t'ai de vred
e"
  220 DATA b$+d$+a$+"p'i de stils
tand"
  230 DATA c$+a$+a$+"t'oeng zjen
gemeenschap met mensen"
  240 DATA a$+a$+d$+"ta joe het b
ezit van het grote"
  250 DATA b$+c$+b$+"tj'ien de be
scheidenheid"
  260 DATA b$+d$+b$+"ju de geestd
rift"
```

```
 270 DATA c$+d$+c$+"swei het nav
olgen"
 280 DATA d$+c$+d$+"koe het werk
 aan het bedorvene"
 290 DATA a$+b$+b$+"lin de toena
dering"
 300 DATA b$+b$+a$+"kwan de besc
houwing"
 310 DATA c$+d$+d$+"sje he het d
oorbijten"
 320 DATA c$+c$+d$+"pi de bekoor
lijkheid"
 330 DATA b$+b$+d$+"po de verspl
intering"
 340 DATA c$+b$+b$+"foe de terug
keer"
 350 DATA c$+d$+a$+"woe wang de
 onschuld"
 360 DATA a$+c$+d$+"ta tsj'oe de
 temmende kracht van het grote"
 370 DATA c$+b$+d$+"i de mondhoe
ken"
 380 DATA d$+a$+c$+"ta kwo het o
verwicht van het grote"
 390 DATA d$+b$+c$+"k'an het onp
eilbare"
 400 DATA c$+a$+d$+"li het zich-
hechtende, het vuur"
 410 DATA b$+a$+c$+"sien de inwe
rking"
 420 DATA d$+a$+b$+"heng de duur
zaamheid"
 430 DATA b$+a$+a$+"t'oen de ter
ugtocht"
 440 DATA a$+a$+b$+"ta tsjwang d
e macht van het grote"
 450 DATA b$+d$+d$+"tjin de voor
uitgang"
 460 DATA c$+c$+b$+"ming i de ve
rduistering van het licht"
 470 DATA c$+c$+a$+"tjia zjen he
t gezin"
 480 DATA a$+d$+d$+"w'ei de tege
nstelling"
 490 DATA b$+c$+c$+'tjien de hin
dernis"
 500 DATA d$+d$+b$+"sie de bevri
jding"
 510 DATA a$+b$+d$+"soen de verm
indering"
 520 DATA c$+b$+a$+"i de vermeer
dering"
 530 DATA a$+a$+c$+"kwai de door
braak"
 480 DATA a$+d$+d$+"w'ei de tege
nstelling"
 490 DATA b$+c$+c$+"tjien de hin
dernis"
 500 DATA d$+d$+b$+"sie de bevri
jding"
 510 DATA a$+b$+d$+"soen de verm
indering"
 520 DATA c$+b$+a$+"i de vermeer
dering"
```

```
 530 DATA a$+a$+c$+"kwai de door
braak"
 540 DATA d$+a$+a$+"kow het tege
moetkomen"
 550 DATA b$+d$+c$+"tsw'ei het v
erzamelen"
 560 DATA d$+c$+b$+"sjeng het om
hoogdringen"
 570 DATA d$+d$+c$+"k'oen de ben
auwenis"
 580 DATA d$+c$+c$+"tjing de wat
erput"
 590 DATA c$+a$+c$+"ke de omwent
eling"
 600 DATA d$+a$+d$+"ting de spij
spot"
 610 DATA c$+d$+b$+"tsjen het op
windende"
 620 DATA b$+c$+d$+"ken het stil
houden, de berg"
 630 DATA b$+c$+a$+"tjien de ont
wikkeling"
 640 DATA a$+d$+b$+"kwei mei het
huwende meisje"
 650 DATA c$+a$+b$+"feng de volh
eid"
 660 DATA b$+a$+d$+"LU de zwerve
r"
 670 DATA d$+c$+a$+"soen het zac
htmoedige, het indringende, de w
ind"
 680 DATA a$+d$+c$+"twei het bli
jmoedige, het meer"
 690 DATA d$+b$+a$+"hwan de oplo
ssing"
 700 DATA a$+b$+c$+"tj'ie de bep
erking"
 710 DATA a$+b$+a$+"tsjoeng foe
innerlijke waarheid"
 720 DATA b$+a$+b$+"siau kwo het
overwicht van het kleine"
 730 DATA c$+c$+c$+"tji tji na d
e voleinding"
 740 DATA d$+d$+d$+"wei tji voor
de voleinding"
1000 FOR f=1 TO 64
1020 READ h$(f)
1030 NEXT f
5000 FOR f=1 TO 64
5010 PRINT h$(f)
5020 NEXT f
7000 SAVE " DATA itjing" DATA h$
(): REM cassette
7010 SAVE *"m";1;" DATA itjing"
DATA h$(): REM microdrive
7030 REM ********************
7050 REM in de SAVE-regels zijn
beide DATA woorden KEYWORDS !!!
```

6.4 Programma DEFKEY verplaatsen

Het nu volgende programme ken van toepassing zijn wanneer U DEFKEY's haaft ingebracht en U wenst tussen de BB code en de DEF KEY's geheugen te reserveren voor bijvoorbeald data of machinecode.

Het programma vraagt U hoeveel ruimte u wilt gaan gebruiken. Vervolgens tikt U op het scherm het aental bytes in. Na het ENTERan vraagt de Spectrum of het door U ingetikte aental correct is. Indien dit het geval is, zullen de DEF KEY's "verhuizen".

```
  10 CLS
  20 REM PROGRAMMA OM RUIMTE TUS
     SEN DE DEF KEY'S EN BB CODE
     TE MAKEN.
  30 PRINT AT 0,0
  40 PRINT "RAMTOP IS NU OP ";DP
EEK(23730)
  50 PRINT

  60 PRINT "BB BEGINT OP    5540
1 ":    REM 55801 VOOR BB 1.8


  70 PRINT
  80 PRINT "HOEVEEL RUIMTE WILT
U TUSSEN DE DEFKEY'S EN BB CODE
 ?"
  90 PRINT FLASH 1;AT 10,0,">";
FLASH 0;"....";AT 10,8,"ENTER";"
 (NA HET INTIKKEN)";CHR$ 11


 100 GET A$: PRINT A$, FLASH 1,"
>": FLASH 0," ";CHR$ 8,CHR$ 8;:
IF A$<>CHR$ 13 THEN GO TO 100
 110 PRINT AT 11,0;" ": ALTER FL
ASH 1 TO FLASH 0
 120 LET A$=""
 130 FOR F=0 TO 5
 140 LET A$=A$+SCRN$(10,F)
 150 IF SCRN$(10,F)=">" THEN PRI
NT AT 10,F;" ": GO TO 170
 160 NEXT F
 170 LET A$=A$( TO LEN A$-1)
 180 IF LEN A$=0 THEN GO TO 90
 190 LET RUIMTE=VAL A$
 200 IF RUIMTE>MEM()-15000 THEN
PRINT AT 15,10; FLASH 1;"TE VEEL
 !!". PAUSE 0. PRINT AT 15,10,ST
RING$(10," "). GO TO 90
 210 PRINT .,"IS DE RUIMTE "; FL
ASH 1,RUIMTE, FLASH 0;" VOLDOEND
E ?";AT 20,0;"(0 = NEE , ENTER =
 JA )"
```

```
  220 GET ANTW
  230 IF ANTW=0 THEN PRINT AT 10,
0;STRING$(384," "): GO TO 20
  240 LET RUIMTE=RUIMTE+1
  250 LET RAMTOP=DPEEK(23730)
  260 DPOKE 23728,RAMTOP
  270 CLEAR RAMTOP-RUIMTE
  280 LET RAMTOP=DPEEK(23728)
  290 LET Z$=MEMORY$()(RAMTOP TO
55400): REM 55800 VOOR 66 1.8
  300 LET NIEUWE RAMTOP=DPEEK(237
30)
  310 LET RUIMTE=RAMTOP-NIEUWE RA
MTOP
  320 POKE NIEUWE RAMTOP,Z$
  330 POKE 55401-RUIMTE,STRING$(R
UIMTE,CHR$ 0)
  340 CLS
  350 PRINT "RAMTOP STAAT NU OP "
,NIEUWE RAMTOP
  360 PRINT
  370 PRINT "PLAATS VRIJ VAN ";(5
  380 PRINT
  390 PRINT "PLAATS BESCHIKBAAR "
;RUIMTE-1
  400 STOP
```

6.5 Programma SPECTRUM ZOEKER

Het volgende programma toont U de snelle en efficiente manier van het POKEn van strings en het nadien weer terugvinden.

Regel 20 reserveert geheugen om de test-string weg te POKEn.

Regel 30 bevat een aantal willekeurig gekozen letters; in het midden is echter het woord SPECTRUM ingebracht. U behoeft de string niet letterlijk in te tikken, als U maar het woord SPECTRUM niet vergeet.

Regel 40 vergroot A$ 60 maal en vervolgens deelt het programma u de LEN mee. Regel 70 DPOKEt de LEN weg en regel 80 POKEt de string weg vanaf adres 40000 (tot en met 40000 + LEN A$).

Regel 90 CLEARt de variabelen en regel 100 geeft de variabele len de waarde van de adressen waar eerder (regel 70) de LEN van A$ werd opgeborgen.

Regel 120 laat A$ gelijk zijn aan MEMORY$ () (adres TO adres-1 + len), oftewel : de inhoud van het geheugen, van 40000 tot 40000-1+ len.

Vervolgens wordt deze (lenge) string gePRINT en wordt U medegedeeld dat er gezocht zal worden naar het woord SPECTRUM.

INSTRING op regel 220 zoekt in het geheugen naar N$ en zorgt ervoor dat veriabele begin het startadres krijgt toegewezen. Indien N$ niet meer voorkomt (variabele begin krijgt de waarde nul) wordt er uit de DO-LOOP lus gesprongen.

TIME$ op regel 320 berekent de tijd die Uw computer nodig had om het woord SPECTRUM terug te vinden.

```
  10 CLOCK 0
  20 CLEAR 39997
  30 LET A$="WererAMW0dffddddffd
fcnvmWDmFmDmFmDmFSPECTRUMREfmfmf
mfmglglgpgoRERRRRcnfjrvEWEWEEWFK
EOZPFLGKRCMERNBRNERYTWEHJERKFLEW
EWWW"
  40 LET A$=STRING$(60,A$)
  50 PRINT AT 10,10;"LEN A$ IS "
;LEN A$
  60 PAUSE 200: CLS
  70 DPOKE 39998,LEN A$
  80 POKE 40000,A$
  90 CLEAR
 100 LET len=DPEEK(39998)
 110 LET adres=40000
 120 LET A$=MEMORY$(adres TO a
dres-1+len)
 130 POKE 23692,255
 140 PRINT A$
 150 PRINT #0;AT 1,0;"HET WOORD
 ""SPECTRUM"" TERUGVINDEN"
 160 PAUSE 50: CLS
 170 LET n$="SPECTRUM"
 180 LET lenn=LEN n$
 190 LET totaal=0
 200 CLOCK "": CLOCK 1
 210 DO
 220 LET begin=INSTRING(adres,ME
MORY$(),n$)
 230 EXIT IF begin=0
 240 LET A$=MEMORY$(begin TO b
egin-1+lenn): LET totaal=totaal+
1
 250 POKE 23692,255
 260 PRINT a$;" adres ";begin;"
tot ";begin+lenn
 270 LET begin=begin+1
 280 LET adres=begin
 290 LOOP
 300 CLS
 310 PRINT ,,a$;" komt ";totaal;
" maal voor"
 320 LET t$=TIME$()
 330 LET tijd=VAL t$(7 TO )
 340 PRINT ,,"in ";tijd;" second
en gevonden"
 350 PRINT ,,"dat is ";totaal/ti
jd;" per seconde"
```

6.6 Programma SCREEN$

Het nu volgende programma tekent een blokjespatroon uit en toont U een van de vele mogelijkheden van ROLL. Na het ROLLen wordt de tekening (geheugenplaatsen 16384 tot 23295, zijnde de DISPLAY FILE en de ATTRIBUTES) opgeslagen in A$.

Regel 330 tot en met 360 POKEt de tekening in stappen van 100 terug. Met deze regels kunt u enigzins experimenteren door bijvoorbeeld de stapgrootte te veranderen en de PAUSE te verkleinen of te vergroten. Het toont U de manier (zoals bij het laden van SCREEN's) waarop de Spectrum de tekening vertreegd weer opbouwt.

TEKENING

```
  10 LET A=10
  20 LET X=0
  30 LET C=180
  40 FOR F=1 TO 8
  50 PLOT (C/4)+X,90
  60 DRAW 0,F*A: DRAW C-(F*2*A),
0: DRAW 0,-F*A
  70 LET X=X+10
  80 NEXT F
  90 PLOT (C/4),90
 100 LET X=0
 110 FOR F=1 TO 8
 120 PLOT (C/4)+X,90
 130 DRAW 0,-F*A: DRAW C-(F*2*A)
,0: DRAW 0,F*A
 140 LET X=X+10
 150 NEXT F
 160 LET A$=MEMORY$() (16384 TO 2
3295)
 170 LET X=30
 180 PAUSE 30
 190 FOR F=1 TO 9
 200 ROLL 9,X-20: ROLL 10,X
 210 ROLL 12,X+20: ROLL 11,X-20
 220 NEXT F
 230 PAUSE 30
 240 CLS
 250 PAUSE 30
 260 POKE 16384,A$
 270 PAUSE 30
 280 CLS
 290 PAUSE 30
 300 FOR F=1 TO 5912 STEP 52
 310 POKE 16384,A$( TO F)
 320 NEXT F
```

6.7 Programma DEF KEY test

Onderstaand programma geeft u een overzicht ven de DEF KEY's. De eerste routine geeft U adressen en lengtes van de DEF KEY's die het eerst zijn ingebracht. Als geen DEF KEY's zijn ingebracht (ramtop = 55400 of 55800) den stopt het programma met de mededeling : "geen defkey aenwezig".

PROC uitlezen geeft U een overzicht van de inhoud van de DEF KEY('s) en PROC totaal uitlezen geeft U een overzicht van de adressen, de karakterkode's en de karakters zelf.

ON ERROR op regel 1 en de routine op regel 220 zorgt ervoor dat bij foutcode 20, 'K Invalid colour' de Spectrum het programma niet onderbreakt, maar de instructie "overslaat" en de volgende instructie efhandelt. De fout ontstaat eventueel wanneer een instructie wordt gegeven om CHR$'s te printen van klaurcode's en dergelijke.

```
  1>ON ERROR 220
 30 LET test=0
 40 LET x=55400: REM 55800 voor
BB 1.8
 50 LET rt=DPEEK(23730)
 60 IF x=rt THEN PRINT "geen de
fkey aanwezig": BEEP 1,1: STOP
 70 LET start=DPEEK(23730)+2: R
EM startadres laatst ingebrachte
 defkey
 80 PRINT : PRINT
 90 PRINT x+1;" start BB code",
,,
100 PRINT start-1;" start DEF K
EYs",,,
110 LET len=DPEEK(start): REM l
en laatst ingebrachte defkey
120 PRINT USING "######";len;" L
EN laatste DEF KEY ",,,
130 LET volgadres=start+len+3
```

```
 140>IF volgadres<x THEN PRINT v
olgadres;" adres voorlaatste DEF
 KEY",,,: LET test=1: REM  adres
 van eventuele voorlaatst ingebr
achte defkey
 150 IF test THEN PRINT USING "#
####";DPEEK(volgadres);" LEN voo
rlaatst ingebrachte       DEF KEY
",,,,
 160 PRINT USING "#####",(x+1)-(
rt);" totaal benodigd voor DEF
      KEY's"
 170 PAUSE 0: CLS
 180 PROC uitlezen
 190 PAUSE 0: CLS
 200 PROC totaal uitlezen
 205 STOP
 210 REM ERROR
 220 IF error=20 AND line=340 TH
EN RETURN : ELSE POP : CONTINUE
 230 DEF PROC uitlezen
 240 FOR a=rt TO x
 250 PRINT CHR$ PEEK a,
 260 NEXT a
 270 PRINT
 280 END PROC
 290 DEF PROC totaal uitlezen
 300 PRINT BRIGHT 1; INVERSE 1;
INK 1;AT 0,0;"adres  karakterkod
e    karakter"
 310 PRINT
 320 LET regel=2
 330 FOR a=rt TO x
 340 PRINT AT regel,0;a;TAB 11;P
EEK a;TAB 22;CHR$ PEEK a
 350 LET regel=regel+1: IF regel
=21 THEN BEEP .1,.1: PAUSE 0: LE
T regel=2: PRINT AT 1,0;STRING$(
640," ")
 360 NEXT a
 370 END PROC
```

6.8 Programma Systeemvariabelen.

Omdat ze zo belangrijk zijn, is het nuttig snel te kunnen beschikken over hun numerieke gegevens in decimaal en hexadecimeel. Terwijl de array met deze gegevens vroeger (71,32) groot was, laat onderstaande toch zien dat dit teveel was.

```
N8   23552 KSTATE 5C00 9    1
N1   23560 LASTK  5C08      2
 1   23561 REPDEL 5C09      3
 1   23562 REPPER 5C0A      4
N2   23563 DEFADD 5C0B 4    5
N1   23565 KDATA  5C0D      6
enzovoort....
```

Het programma systeemvariabelen maakt nu deel uit van de 'directory' van het Z-80 Spectrum-systeem, gezien het beschikt over de array S$ - die alle gegevens in min of meer gecodeerde vorm bevat : s$ (f).

(1 TO 1) = N	Het POKEn van de sysvar met N heeft geen blijvend effect.
(1 TO 1) = X	POKEn van deze systeemvariabele is gevaarlijk of althans niet al te eenvoudig.
(2 TO 3) =	Aantel adressen van de systeemvariabele.
(4 TO 9) =	Naam van de systeemvariabele.
(10 TO 11) =	Het adres van de systeemvariebele of het eerste adres van de systeem variabele.

De data worden nu ingebracht - en hiervoor gebruikt u de datalijst die volgt.

```
 llist  datalijst  'sysvardata'

    1000  DATA "N8 9  KSTATE"
    1010  DATA "N1    LASTK"
```

Pas achteraf zetten we met enkele Basicregels in iedere s$ het edres van de systeemvariabele, in de vorm van een CHAR$(),

```
    8000 DIM S$(71,11): RESTORE
    8010 LET ADRES=23552
    8020 FOR F= 1 TO 71 : READ D$
    8030 LET S$(F, TO 9)= D$
    8040 LET S$(F, 10 TO )=CHAR$(ADRES)
    8050 LET ADRES=ADRES+ VAL S$(F,2 TO 3)
    8060 NEXF F : STOP
```

Het programma geeft listings van de systeemvariabelen via keuze 1 (op orde van adres) en via keuze 2 (op alfabetische orde). De belangrijkste keuze is 3: het opzoeken van 1 of meer systeem variabelen. U INPUT de naam van de te zoeken sys. variabele Dit hoeft niet da volledige naam te zijn; de eerste letters volstaan. Daarna wordt de mogelijkheid gaboden
de inhoud van de gezochte systeemvariabele te wijzigen. Probeer dit maar eens met bv. ATTRP en POKE deze met bv. 78. U ziet onmiddellijk het resultaat. Er blijven ongeveer 26700 bytes vrij na het doorlopen van alle mogelijkheden (disc-omstandigheden).

Net als het programma 'var-info' en 'meminfo' is dit een hulp-programma dat achteraan in andere programma's zou kunnen staan. Het kan den in zijn geheel in een DEF PROC worden gezet.

6.8 LISTING 1: DATA SYSTEEMVARIABELEN

```
501 DATA "N8 KSTATE"
502 DATA "N1 LASTK "
503 DATA " 1 REPDEL"
504 DATA " 1 REPPER"
505 DATA "N2 DEFADD"
506 DATA "N1 KDATA "
507 DATA "N2 TVDATA"
508 DATA "X38STRM5 "
509 DATA " 2 CHARS "
510 DATA " 1 RASP  "
511 DATA " 1 PIP   "
512 DATA " 1 ERRNR "
513 DATA "X1 FLAGS "
514 DATA "X1 TVFLAG"
515 DATA "X2 ERRSP "
516 DATA "N2 LISTSP"
517 DATA "N1 MODE  "
518 DATA " 2 NEWPPC"
519 DATA " 1 NSPPC "
520 DATA " 2 PPC   "
521 DATA " 1 SUBPPC"
522 DATA " 1 BORDCR"
523 DATA " 2 EPPC  "
524 DATA "X2 VARS  "
525 DATA "N2 DEST  "
526 DATA "X2 CHANS "
527 DATA "X2 CURCHL"
528 DATA "X2 PROG  "
529 DATA "X2 NXTLIN"
530 DATA "X2 DATADD"
531 DATA "X2 ELINE "
532 DATA " 2 KCUR  "
533 DATA "X2 CHADD "
534 DATA " 2 XPTR  "
535 DATA "X2 WORKSP"
536 DATA "X2 STKBOT"
537 DATA "X2 STKEND"
538 DATA "N1 BREG  "
539 DATA "N2 MEM   "
540 DATA " 1 FLAGS2"
541 DATA "X1 DFSZ  "
542 DATA " 2 S-TOP "
543 DATA " 2 OLDPPC"
544 DATA " 1 OSPPC "
545 DATA "N1 FLAGX "
546 DATA "N2 STRLEN"
547 DATA "N2 TADDR "
548 DATA " 2 SEED  "
549 DATA " 3 FRAMES"
550 DATA " 2 UDG   "
551 DATA " 1 COORDX"
552 DATA " 1 COORDY"
553 DATA " 1 PPOSN "
554 DATA " 1 PRCC  "
555 DATA " 2 ECHOE "
556 DATA " 2 DFCC  "
557 DATA " 2 DFCCL "
558 DATA "X1 SPOSUC"
559 DATA "X1 SPOSUL"
560 DATA "X2 SPOSLC"
561 DATA " 1 SPOSL "
562 DATA " 1 SCRCT "
563 DATA " 1 ATTRP "
564 DATA " 1 MASKP "
565 DATA "N1 ATTRT "
566 DATA "N1 MASKT "
567 DATA " 1 PFLAG "
568 DATA "N30MEMBOT"
569 DATA " 2 NONAME"
570 DATA " 2 RAMTOP"
571 DATA " 2 PRAMT "
```

6.8 LISTING 2: CONVERSIEREGELS

```
CONVERSIEREGELS

1000 LET ADRES=23552
1005 RESTORE
1010 DIM S$(71,11)
1020 FOR F=1 TO 71
1030 READ D$
1040 LET S$(F, TO 9)=D$
1050 LET S$(F,10 TO )=CHR$(ADRE
S)
1060 LET ADRES=ADRES+VAL S$(F,2
TO 3)
1070 NEXT F: STOP
```

6.8 LISTING 3: PROGRAMMA

```
SYSTEEMVARIABELEN

8000 DEF PROC SYSVAR
8010 PROC SVMENU: STOP
8020 DEF FN G(F)=VAL S$(F,2 TO 2
)
8030 DEF FN B(F)=NUMBER(S$(F,10
TO ))
8040 DEF PROC SVN: SORT S$() (10
TO ): PROC END: END PROC
8050
8060 DEF PROC SVA: SORT S$() (4 T
O ): PROC END: END PROC
8070
8080 DEF PROC SVAL: CLS : PRINT
'A$(3)'': PROC SVA: PROC LUS: PR
OC END: END PROC
8090
8100 DEF PROC SVNL: CLS : PRINT
'A$(4)'': PROC SVN: PROC LUS: PR
OC END: END PROC
8110

8120 DEF PROC LUS: FOR F=1 TO 71
: PROC PRSO: NEXT F: PROC END: P
ROC SVMENU: END PROC
8130
8140 DEF PROC END: PRINT '" TAAK
 VOLBRACHT ": PAUSE 40: IF I<2 T
HEN PROC SVMENU: ELSE : END PROC
```

```
8150 DEF PROC ZOEKSV: CLS : INPU
T "GEEF DE NAAM "+CHR$ 13; LINE
i$: FOR F=1 TO 71: IF INSTRING(4
,s$(F),I$)=4 THEN PROC PR2
8160 NEXT F: PROC END
8170 LET I$=I$+" DPOKE OR POKE "
: PROC JN: PROC POKE: END PROC
8180 DEF PROC POKE: DO : INPUT "
 INPUT HET  SYSVAR NUMMER RECHTS
 ";I: EXIT IF STR$ I>"0" AND STR
$ I<"72"
8190 LOOP : LET F=I

8200 IF FN G(F) THEN INPUT ( BRI
GHT 1;" POKE ";S$(F,4 TO 9)+" ME
T: ");P. POKE FN B(F),P
8210 IF FN G(F)=2 THEN INPUT ( B
RIGHT 1;" DPOKE ";S$(F,4 TO 9)+"
 MET: "),P: DPOKE FN B(F),P
8220 LET F=I: PROC PR2: LET I$="
 NOG EEN ?": PROC JN: PROC POKE
8230 END PROC
8240 DEF PROC PR2: PRINT '': PRO
C PR SV: IF NOT FN G(F)-1 THEN P
RINT " PEEK =";PEEK FN B(F)
8250 IF FN G(F)=2 THEN PRINT " D
PEEK()=";DPEEK(FN B(F))
8260 END PROC
8270 DEF PROC SVMENU: DIM A$(5,1
0)
8280 LET A$(1)=" SORT PER ADRES"
8290 LET A$(2)=" SORT OP  NAAM"
8300 LET A$(3)=" LIST ALFANUM."
8310 LET A$(4)=" LIST NUMERIEK"
8320 LET A$(5)="OPZOEKEN"
8330 CLS : PRINT '" Z80 SYSTEEM
VARIABELEN "'': FOR F=1 TO 5: PR
INT  " ";F-1;" ";A$(F): NEXT F: P
RINT STRING$(2,CHR$ 13)+" 5 STOP
 "+CHR$ 13+" 6 RETURN "
8340 DO : GET I: IF NOT I-5 THEN
 STOP : ELSE EXIT IF I<7 AND I>=
0
8350 LOOP
8360 IF NOT I THEN PROC svn: ELS
E IF NOT i-1 THEN PROC sva: ELSE
 IF I=2 THEN PROC SVAL: ELSE IF
I=3 THEN PROC SVNL: ELSE IF I=4
THEN PROC zoeksv: ELSE CLS : PRI
NT "NAAR UW HOOFDPROGRAMMA": STO
P
8370 END PROC
8380 DEF PROC PRSV: POKE 23692,2
55  PRINT " ";S$(F, TO 3);"  "; F
N B(F);" ";S$(F,4 TO 9);" ";HEX$
(FN B(F))+"       ";F: END PROC
```

```
8390 DEF PROC JN: DO : INPUT (I$
+"(J/N) "+CHR$ 13); LINE J$: IF
J$="N" THEN PROC SVMENU: ELSE :
EXIT IF J$="J"
8400 LOOP : END PROC
8410 STOP
8420 DEF PROC SAVESV
8430 SAVE *"M";1;"SYSVAR CODE DA
TA " DATA S$(): CLEAR : SAVE *"M
";1;"LASTSYSVAR" LINE 8450
8440 STOP
8450 LOAD *"M";1;"SYSVAR CODE DA
TA " DATA S$(): STOP
8460 PRINT AT 0,25;LINE: RETURN
8470 END PROC
```

6.9 PROGRAMMA ADRESSENBESTAND

ALGEMENE BESCHRIJVING

Dit programma laat U toe een 150-tal adressen te selecteren en te manipuleren. Het is zodanig geschreven dat op eenvoudige wijze alle rubrieken en veldlengten aan Uw eigen wensen zijn aan te passen.

In het hierbij afgedrukte programma is thans nog circa 7Kb vrij, zodat het aantal adressen ook verhoogd kan worden of er kunnen LPRINT procedure's toegevoegd worden , zodat de geselecteerde adressen op etiketten of lijsten geprint kunnen worden.

De diverse VAL "waarde" hebben ertoe bijgedragen dat er zo weinig mogelijk geheugenruimte wordt gebruikt. Bovendien werd gebruik gemaakt van de JOIN opdracht om regels aan elkaar te koppelen.

Regel 20 van het programma roept het MENU op, van waaruit U in het begin (als nog geen DATA ingevoerd zijn), altijd KEUZE 1 geeft, tenzij U de DATA van de microdrive of cassette gaat halen, dan is het KEUZE 3. De SAVE en LOAD routine's zijn in vier afzonderlijke listings opgenomen, te weten voor:
BETA BASIC 1.8 , BETA BASIC 1.9 , CASSETTE en MICRODRIVE(OPUS)

Het aanroepen van het MENU heeft de eliereerste keer tot gevolg dat regels 1510 en 2500 tot 2510 gewist(geDELETE) worden. Het is wellicht raadzaam om bij het intikken van regel 2510 eerst een REM (zoals in de listing) voor de opdrecht te zetten en die er pas achteraf ALS HET PROGRAMMA FOUTLOOS LOOPT weer uit te halen.

Naast de listing hebben we een listing met ALLE VARIABELEN , CONSTANTEN en PROCEDURES opgenomen. Alsmede FLOWCHARTS van het HOOFDMENU en de HOOFDMENUKEUZES.

HET INTIKKEN

Om het programma te kunnen intikken dient U in het bezit te zijn van BETA BASIC 1.8 of 1.9.
Laadt eerst de BETA BASIC + BB CODE . Na het laden dient eerst de RAMTOP verlaagd te worden door:

DPOKE 23730,55300 of: CLEAR 55300

U kunt het effekt PEEKen door: PRINT DPEEK (23730)
U krijgt dan als antwoord : 55300

Tik nu het programma in. (voor cassette, microdrive, BB 1.8 en BB 1.9 zijn verschillende procedures gemaakt, zie daarvoor de aparte listings achter de listing van het programma.)
Nadat het volledige programma is ingetikt EERST regel 2510 met een REM voorzien. SAVE het programma en de BB door als DIRECTE OPDRACHT te geven: PROC saveBB <ENTER>.
Daarna VERIFY op de gebruikelijke manier.

Als alles OK is kunt U nu GO TO 1 geven , waarna we het MENU op het scherm krijgen. Vermijdt het gebruik van RUN , omdat dit alle variabelen en DATA zal CLEARen. Het programma is volledig MENU-gestuurd.

BESCHRIJVING VAN HET GEBRUIK

KEUZE 1 VAN HET HOOFDMENU

Bij deze keuze wordt er U eerst gevraagd of eventuele aanwezige data gewist mogen worden uit het interne geheugen.

SUBKEUZES:

1: Als U 1 geeft worden de eventuele aanwezige data gewist en loopt het programma verder en kan de invoer beginnen.
2: Als U 2 geeft worden de nieuwe data TOEGEVOEGD aan de bestaande gegevens. U krijgt dan een nieuw keuze scherm , waarvan U eerst rubriek 1 dient op te geven.
s: Toetst U de kleine letter s in dan keert U terug naar het HOOFDMENU.

Na het invoeren van rubriek 1 (zie SUBKEUZE 2) verandert het scherm enigszins. U bent nu in de gelegenheid om SUBRECORD per SUBRECORD in de door U gewenste volgorde in te vullen.
Indien U bijvoorbeeld rubriek 3 wenst in te vullen en deze rubriek was al ingevuld, dan kunt U als invoer * geven. Het programma zorgt er dan voor dat de eerder ingevoerde data NIET overschreven worden.
De in te vullen rubrieken zijn genummerd van 1 t/m 8 en komen overeen met de nummering op het scherm.
INPUT 9 heeft tot gevolg dat het record zoals het op het scherm wordt getoond, in het intern geheugen wordt opgeslagen en dat een volgend record kan worden ingevoerd.
Bij toe te voegen data wordt er gecontroleerd of de data er nog bij kunnen. Is dit niet het geval dan wordt automatisch teruggegaan naar het HOOFDMENU.

KEUZE 2 VAN HET HOOFDMENU

Deze keuze stelt U in staat de ingevoerde of ingelezen data op verschillende manieren te selecteren en manipuleren.
Het scherm is vrijwel identiek aan die van KEUZE 1.

SUBKEUZES:

KEUZE A: U kunt nu een combinatie maken van de rubrieken 1 t/m 8, dus bijvoorbeeld NAAM en WOONPLAATS en TELEFOONNUMMER.
Rubriek 9 heeft tot gevolg dat de selectie-criterie worden afgebroken en dat er een print-keuze gemaakt moet worden. Later meer hierover.
U kunt in keuze A ook nogmaals (na de selecties) keuze A geven (van tot) ; bijvoorbeeld:

```
rubriek 2 naam: janssens
rubriek 4 PTT : 1056
KEUZE A
rubriek 3 straat van: x tot: y
```

U krijgt dan alle janssens met PTTnummer 1056 in de straten die tussen x en y liggen.
Op deze wijze is het mogelijk om op alle subrecords (fields) te selecteren met als optie: (van-tot) te sorteren.

KEUZE B: Stelt U in staat een enkel record van .... tot .... te selecteren. Bijvoorbeeld alle records van PTTnummer 1000 tot 1500.

KEUZE C: Stelt U in staat om per subrecord alfabetisch te selecteren. Deze procedure laat heel duidelijk zien hoe krachtig , snel en efficient gesorteerd kan worden met BETA BASIC.

KEUZE D: Stelt U in staat om op een deel(tje) van de records te selecteren (huisnummers,netnummers,delen van naam, enz.)

KEUZE 2 VAN HET HOOFDMENU,vervolg.

KEUZE 1 tot 8:
Stelt U in staat alle subrecords van de ingetoetste rubriek (1/8) direct te zoeken. Alle keuzes werken op dezelfde wijze.
KEUZE 9:Terug naar het HOOFDMENU.

PRINTKEUZES:

Nadat de invoer ingebracht is dient U de printkeuze op te geven. Er zijn twee mogelijkheden.

PRINTKEUZE 1:
Geeft een totaaloverzicht van alle gevonden records die aan de selectie-criteria voldoen. (Het programme in zijn huidige vorm kan 99 records voor dit totaaloverzicht bevatten, indien U dat wenst is dit eenvoudig aan te passen door RAMTOP bijvoorbeeld op 55250 te zetten, evenals de bijbehorende variabele rt. Voor Bete Basic 1.8 is dit niet noodzakelijk omdat de BB CODE op adres 55800 begint en er dus geen gevaar voor overschrijven bestaat.)
Tijdens het printen kan er teruggekeerd worden naar het zoekmenu door op 's' te drukken. Het scherm toont maximaal 21 records (volgnummer,naam en subrecord waarop geselecteerd werd.), waar U dan een of meerdere records kunt uitlichten, door het volgnummer in te toetsen.

In plaats van een volgnummer kunt U ook ENTER geven (volgende bladzijde, als die er is.) of * (terug naar het zoekscherm.).
Heeft U een volgnummer ingetoetst dan komt het record op het scherm zoals U het heeft ingevoerd.
Nu kunt U:

MUTEREN (wijzigingen aanbrengen)	(m)
Terug naar ZOEKMENU	(s)
VERWIJDEREN	(v)
Terug naar TOTAALOVERZICHT	(ENTER)

Als U 'm' intoetst dan ontstaat de mogelijkheid om alle subrecords te wijzigen door het betreffende rubrieknummer in te toetsen. De mode waar U nu in bent is dezelfde als de invoer en toevoeg mode.
Als U 's' intoetst dan krijgt U de volgende statistieken:
DATA FOUND = aantal gevonden records naar selectie.
DATA READ = aantal ingelezen records
DATA DELETE = aantal afgevoerde records
DATA CHANGED = aantal gewijzigde subrecords
Hierna keert U weer terug naar het ZOEKMENU.

Als U 'v' intoetst dan wordt het record op het scherm afgevoerd , nadat dit door U met 'j' is bevestigd.
Als U 'ENTER' intoetst dan kunt U een volgend record uit het totaaloverzicht lichten.

PRINTKEUZE 2:
Geeft de records afzonderlijk weer, in de volgorde zoals ze werden ingevoerd. U heeft daarbij dezelfde SUBKEUZES als bij PRINTKEUZE 1 (m, s, v, ENTER). Tijdens het zoeken kunt U door 's' in te toetsen terugkeren naar het zoekmenu.

KEUZE 3 VAN HET HOOFDMENU:

Nu bent U in het SAVE en LOAD MENU.
De volgende mogelijkheden

KEUZE 1: CLEARt alle data en variabalen en SAVEt het programma en de BB CODE.
KEUZE 2: Vreagt om een bestandsnaam en OPENt een FILE en schrijft de ingevoerde data weg.(cassette,microdrive of disc)
KEUZE 3: Vraagt evanaans om da naam van het bestand en OPENt een (eerder beschreven) FILE en leest de data in en zet da telvariabele (j) gelijk aan het aantal ingelezen records
KEUZE 4: Brengt U terug naar het HOOFDMENU, als U per abuis in dit menu bent terechtgekomen.

Bij de keuzes 1, 2 en 3 komt U eutomatisch terug in het hoofdmenu na het beeindigen van de SAVE en LOAD opdrachten.

N.B. Keuzes 2 en 3 voor cassette OPENen GEEN FILE maar schrijvan de gehela ARRAY weg naar cassetta.

KEUZE 4 VAN HET HOOFDMENU:

Keuze 4 onderbreekt het programma. U kunt het programma weer opstarten met de DIRECTE OPDRACHT: GO TO 1 of met PROC MENU.

Deza directe opdracht kunt U ook gebruiken na een TAPE LOADING ERROR of MICRODRIVE FOUT.
U dient enkel RUN te gebruiken als het geheugen gewist mag worden.

BESCHRIJVING VAN DE GEBRUIKTE BETA BASIC:

Hat programma is voornamelijk opgebouwd uit lossa procedures. De meeste procedures zijn echter NIET AFZONDERLIJK te gebruiken, omdat ze vanuit de procedure andere procedures aanroepen.

De functie INSTRING wordt meerdere malen gebruikt om te zoeken naar een bepaalde string in de dete (zie regel 320 t/m 340).

Da functie STRING$ wordt gebruikt om bepaelde delen van het scherm (op) te vullan met spaties. Voordeel is det niet steeds het gehele scherm geCLSt en weer opgebouwd behoeft te worden.
PROC CLEAR is hiervan een voorbeeld.

GET wordt gebruikt om op een snelle en korte wijze aan keuze te maken uit een MENU.
ELSE is een uitbreiding van de IF ... THEN ... instructie,waardoor een tweede IF / THEN overbodig wordt.
SORT zorgt voor een zear snalle sortering ven de DATA per rubriek in PROC SORT.
USING formateart getallen zodanig det ze 'netjes' onder elkaer komen te steen in PROC datetel.
DO UNTIL EOF ..... LOOP is een lus die herhaeld wordt net zoleng totdat de waarde ven EOF gelijk aan 1 wordt.
Zie PROC load (versie 1.9) voor het gebruik van deze lus.

DPEEK wordt o.a. gabruikt in PROC saveBB om RAMTOP (=rt) op de juiste waarde te zetten.

GO TO ON wordt in het de menu's gebruikt om het programma neer de gewenste regelnummers te sturen. (voorafgegaan door GET).

6.9 OVERZICHT GEBRUIKTE VARIABELEN,CONSTANTEN,STRING$,PROCEDURES

NUMMERIEKE VARIABELEN:

a , bepaalt welke procedure wordt aangeroepen vanuit het SAVE en LOAD menu.
aa, waarde die bepaalt welke waarde in PROC keuze toegekend moet worden.
angle, telt in PROC keuze2 het aantal malen dat gaENTERt werd.
ch, teller van het aantal gewijzigde records.
co, teller van aantal gevonden records tijdens zoekacties.
del,teller van het aantal gewiste records .
hmenwkeus,bepaalt welke procedure wordt aangeroepen vanuit het hoofdmenu.
j, teller van het aantal ingevoerde records.
ke, waarde die bepaald wordt na invoer van gecombineerde selecties.
kl, hulpvariabele in PROC cijfer.
km, hulpvariabele in PROC cijfer.
p, hulpvariabele in PROC pk
pk, teller van het aantal gevonden selecties ten behoeve van PROC print2.
pke,teller ten behoeve van PROC print2 in PROC pk.
test,bepaalt welke printwijze toegepast moet worden.
wb, waarde die bepaald wordt na invoer van een selectie-keuze.
x, slicer van b$ in PROC ingave en keuze.
xos,BETA BASIC variabele, altijd aanwezig.
xrg,BETA BASIC variabele, altijd aanwezig.
xx, slicer van de b$ in PROC ingave en keuze.
y, slicer van de b$ in PROC ingave en keuze.
yos,BETA BASIC variabele altijd aanwezig.
yrg,BETA BASIC variabele altijd aanwezig.
yy, slicer van de b$ in PROC ingave en keuze.

NUMMERIEKE CONSTANTE:

cijf,controller in PROC cijfer
kk, printpositie in PROC print2.
x1, b$ array index, waarde 1.
x10,b$ array index, waarde 67
x11,b$ array index, waarde 68
x12,b$ array index, waarde 79
x13,b$ array index, waarde 80
x14,b$ array index, waarde 95
x15,b$ array index, waarde 96
x16,b$ array index, waarde 100
x17,bevat de waarde van adres 23658, om CAPS LOCK aan (8) of uit (0) te zetten.
x2, b$ array index, waarde 4
x3, b$ array index, waarde 5
x4, b$ array index, waarde 25
x5, b$ array index, waarde 26
x6, b$ array index, waarde 46
x7, b$ array index, waarde 47
x8, b$ array index, waarde 51
x9, b$ array index, waarde 52
z, printpositie in PROC print

FOR / NEXT VARIABELEN:

k, lus in PROC savedata en loaddata voor het SAVEn en LOADen van de b$ array.
m, diverse procedures, bevat de waarde van het b$(m) record, nodig voor het zoeken en printen van date.
v, lus in PROC cijfer ter controle van de input.
w, lus in PROC pk ter controle van data in PROC print2.

6.9 OVERZICHT GEBRUIKTE VARIABELEN,CONSTANTEN,STRINGS,PROCEDURES

STRING ARRAYs :

- b$(150,100), string die de ingevoerde data bevat.
- p$(11,8) , string met constanten die de rubriek-gegevens bevat.

STRING VARIABELEN.

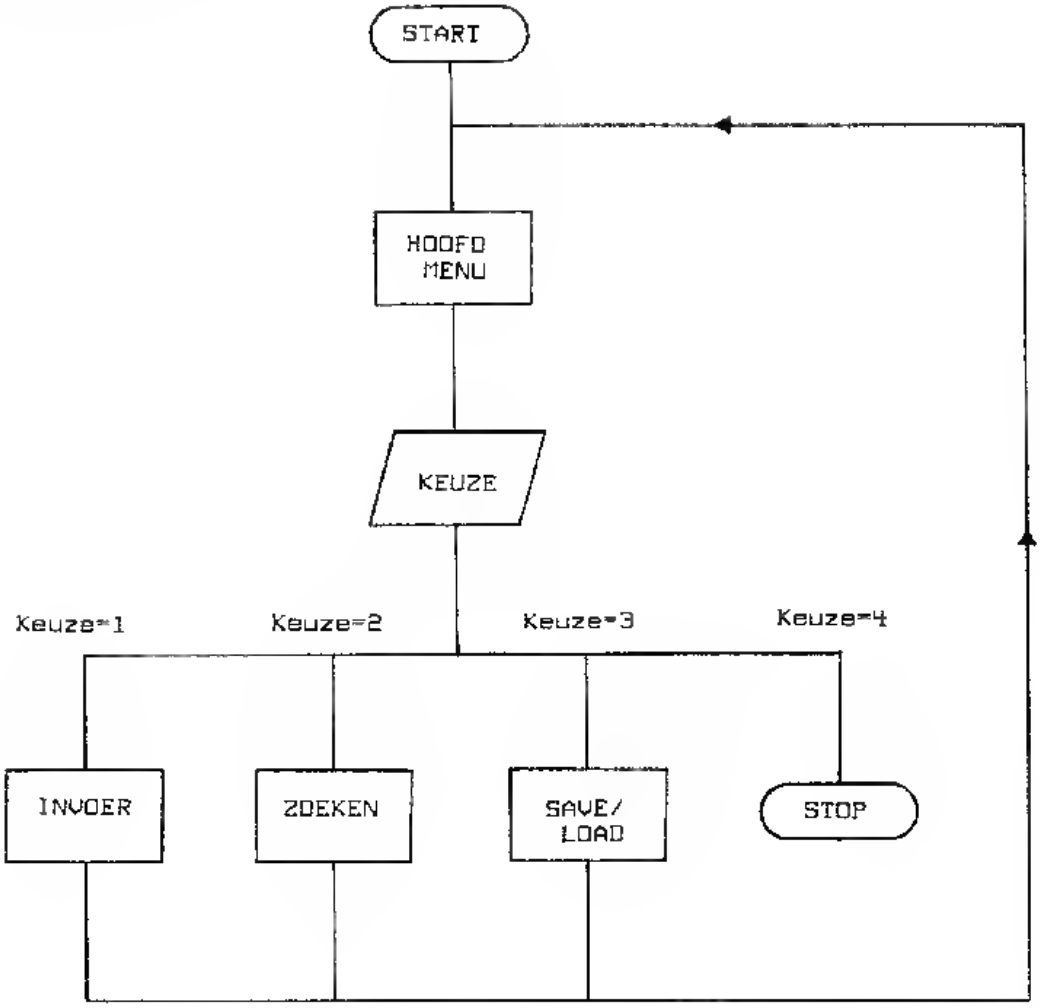
- a$, hulpstring
- c$, hulpstring
- d$, hulpstring
- f$, hulpstring
- g$, hulpstring
- h$, hulpstring
- i$, hulpstring
- k$, hulpstring
- l$, hulpstring
- n$, bevat de te zoeken string in PROC zoek.
- o$, bevat de waarde van een geselecteerd record in PROC keuze2.
- v$, bevat de te zoeken beginstring in PROC vantot.
- w$, bevat de te zoeken eindstring in PROC vantot.
- x$, bevat antwoord in PROC tussen.
- e$, bevat spaties om delen van het scherm te wissen.

PROCEDURES:

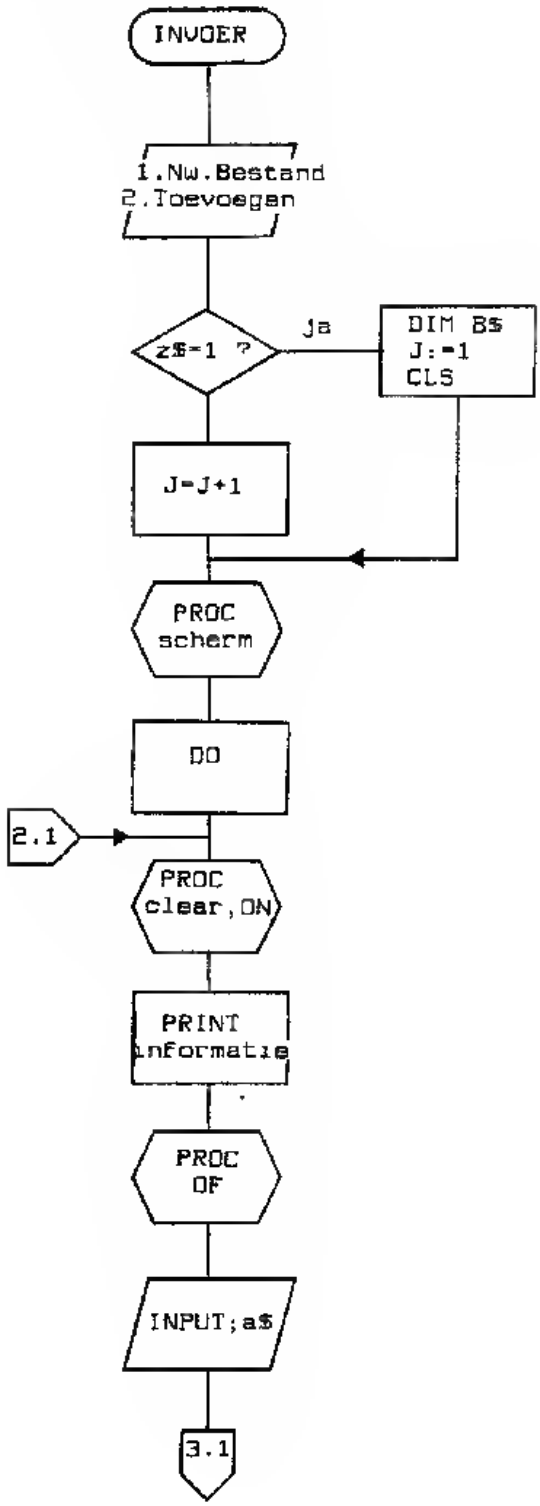
- alfa , vraagt U welke rubriek gesortaard moet worden.
- and , controleert of de gevraagde data aanwezig is in array b$.
- cijfer , controleert een deel van de invoer bij PROC keuze2.
- clear , wist een deel van het scherm.
- clear2 , wist een ander deel van het scherm.
- datatel, print de totalen van de statistieken.
- ingave , invoer en wijzigen van data.
- inputvar,leest de rubriekgegevens in, wordt na het inlezen geDELETEt.
- instr , stelt U in staat een deel(tje) van de ingevoerde data terug te vinden.
- invoer , verzorgt de invoer van data in het bestand.
- keuze , kent waarden toe aan de keuza-input, afhankelijk van waar deze PROCedure werd aangeroepen.
- keuze2 , vraagt U na PROC print2 wat U verder wilt.
- leeg , de hulpstring$ worden op LEN nul gezet.
- loaddata,OPENt een file en leest een eerder beschreven bestand in.
- menu , roept het HOOFDMENU op.
- niet , geeft CLS als test=1.
- off , zet BRIGHT, INVERSE en INK op nul (0).
- on , zet BRIGHT en INVERSE op EEN (1).
- pk , teller van het overzicht van PROC print2.
- poke , POKEt de b$(m)-waarden weg om later t.b.v. PROC print2 weer gePEEKt te worden.
- print , printen van gevonden data,per afzonderlijk record.
- print2 , geeft een overzicht van geselecteerde records.
- save88 , CLEARt de variabelen en SAVEt het programma en de Beta Basic.
- savedata,OPENt een file en beschrijft het met de ingevoerde data.
- saveload,submenu voor het SAVEn en LOADen van programma en data.
- scherm ,printen van de rubrieken op de juiste plaatsen.

PROCEDURES:
(VERVOLG) selectie,invoerkeuze van 1 t/m 8 (al dan niet gecombineerde) RUBRIEKEN.
sort ,sorteert op de door U opgegeven rubrieken alle records.
test ,er wordt verzocht een printkeuze te maken.
tussen ,geeft U de mogelijkheid om tussen de gevonden records een bepaalde keuze te maken.
vantot ,bepaalt het zoeken van --- tot, bijv: naam a tot naam h
varia ,zet diverse variabelen op default .
verwijder,verwijdert het door U opgegeven record uit het bestand.
zoeken ,invoer van de te zoeken (sub)records.

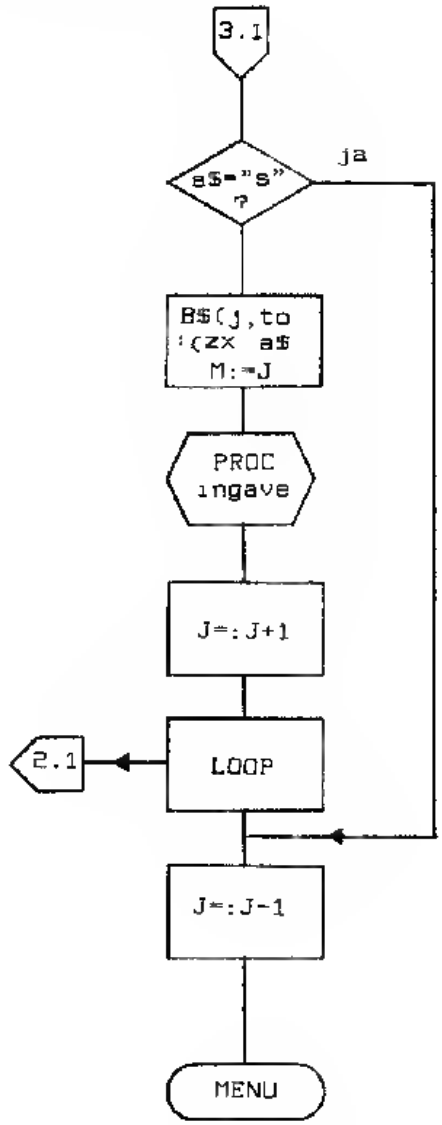
HOOFDMENU
SYSTEEM FLOWCHART



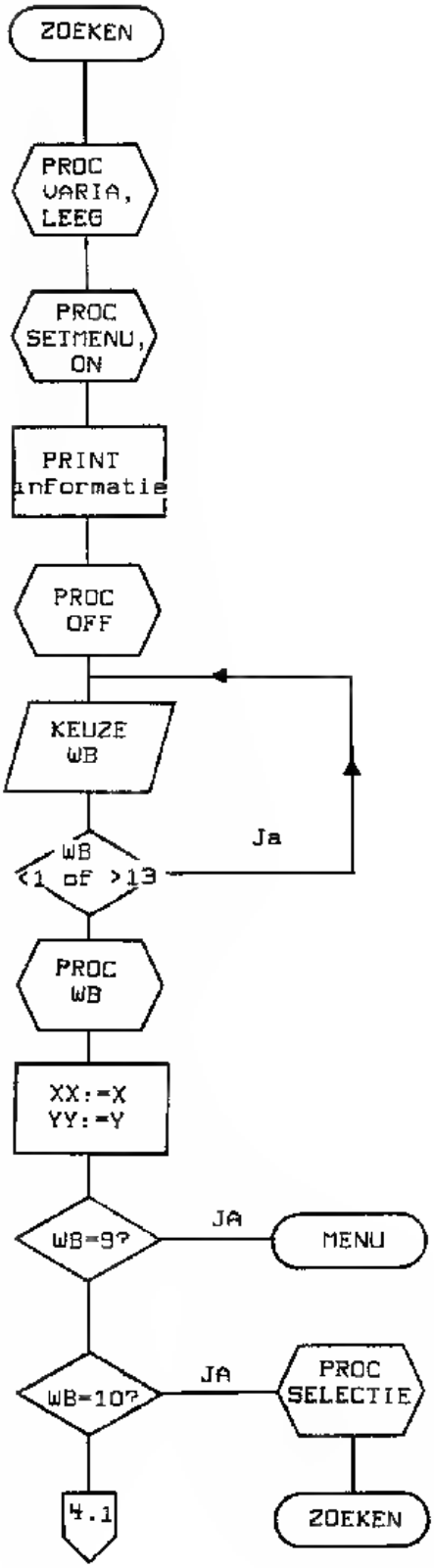
HOOFDMENUKEUZE 1
PROCEDURE FLOWCHART



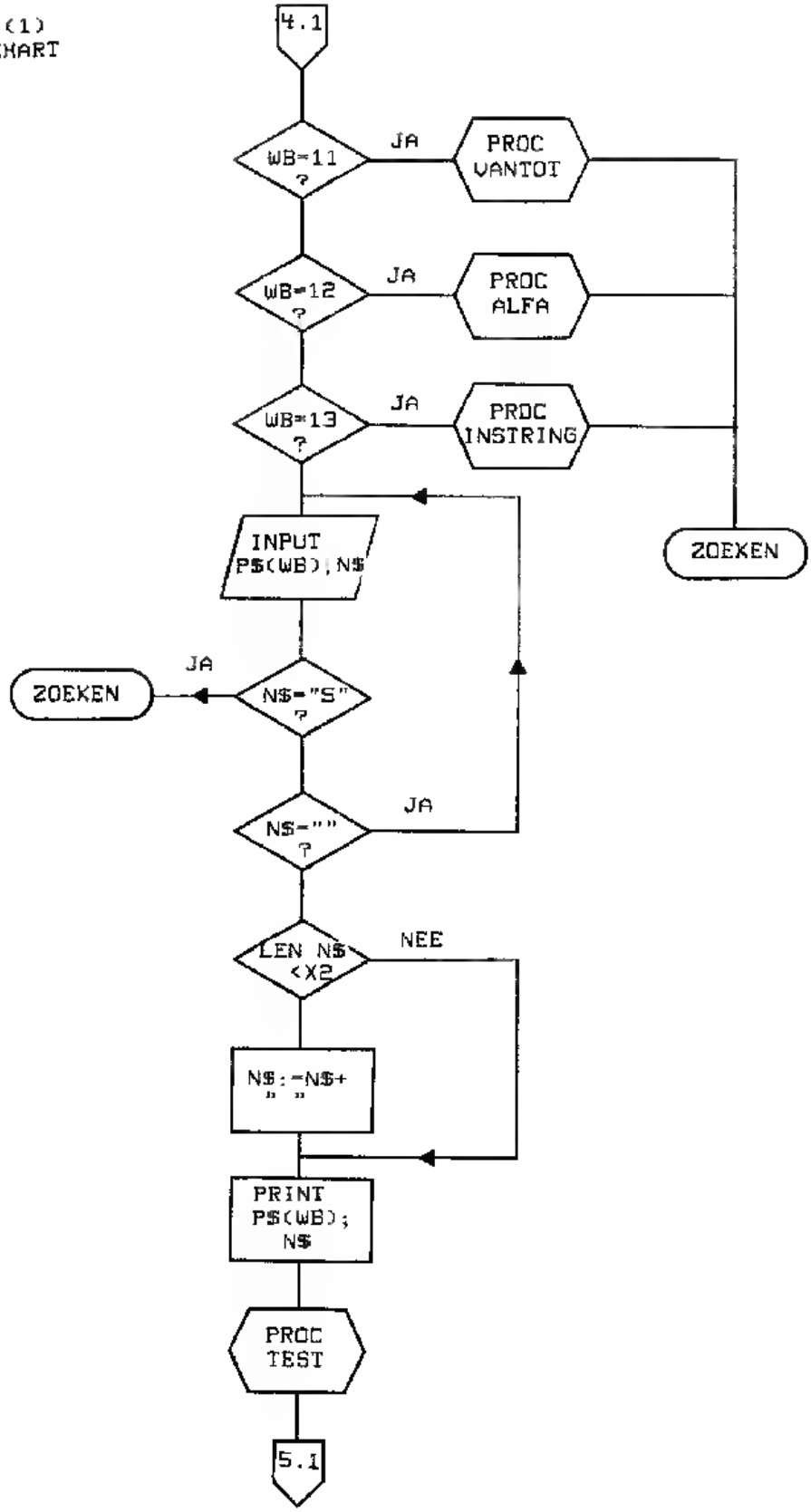
vervolg invoer
PROCEDURE FLOWCHART



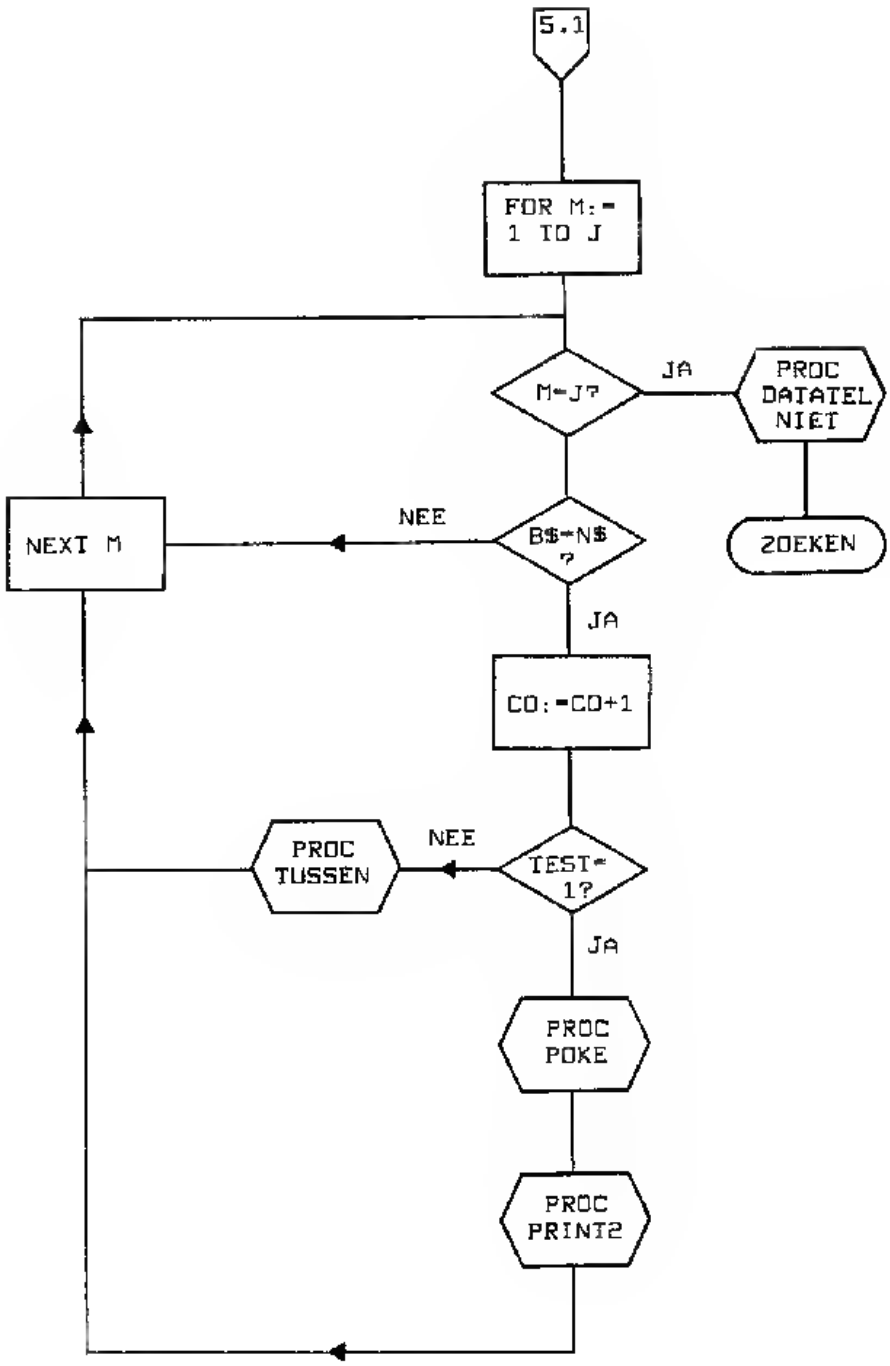
HOOFDMENUKEUZE 2
PROCEDURE FLOWCHART



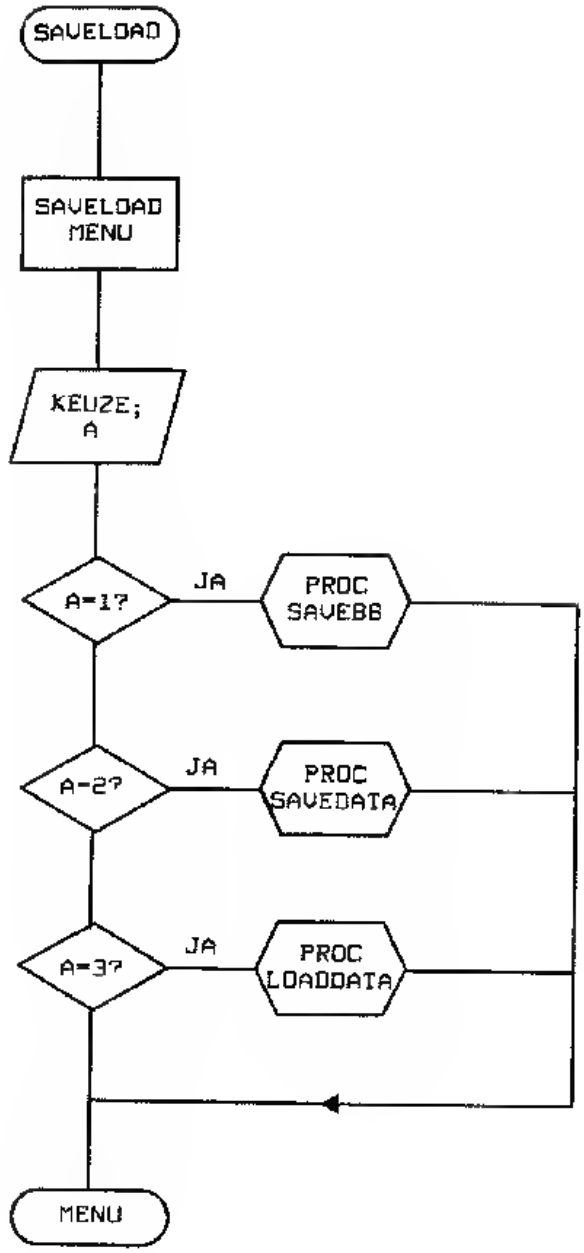
vervolg zoeken (1)
PROCEDURE FLOWCHART



vervolg zoeken (2)
PROCEDURE FLOWCHART



HOOFDMENUKEUZE 3
PROCEDURE FLOWCHART



```
   10 REM 20 juli 1985 s.girard
   20 PROC menu
   30 STOP
   40 DEF PROC invoer
   50 CLS : PRINT AT VAL "10",VAL
 "0";"1 DATA nieuw bestand",,,"2
  DATA toevoegen": GET z$: CLS
   60 IF z$="1" THEN DIM b$(VAL "
150",x16): LET j=x1: CLS : ELSE
LET j=j+1
   70 PROC scherm
   80 DO
   90 PROC clear
  100 PROC on: PRINT INK VAL "1";
AT VAL "18",VAL "0","Voer s in o
m te stoppen": PROC off: PRINT I
NVERSE VAL "1"; INK VAL "3",AT V
AL "0",VAL "0"; ITEMNR   "; INVE
RSE VAL "0"; INK VAL "1";" ";j
  110 INPUT (p$(VAL "1")); LINE a
$
  130 EXIT IF a$="s" OR j>149
  140 LET b$(j, TO x2)=a$: PRINT
AT VAL "2",z,b$(j, TO x2): PRINT
  AT VAL "18",VAL "0";STRING$(VAL
  "32",e$)  LET m=j: PROC ingave
  150 LET j=j+VAL "1"
  160 LOOP
  175 LET j=j-1
  180 PROC menu
  190 END PROC
  200 DEF PROC zoeken: PRINT AT V
AL "0",VAL "0",STRING$(VAL "32",
e$)
  210 CLS
  220 PROC varia: PROC leeg: PROC
  scherm: PROC on
  230 PRINT INK VAL "4";AT VAL "1
7",VAL "0","A=combinatie B=van-t
ot C=alfa   "; INK VAL "3";"1=kl
nr        2=naam     3=straat "; I
NK VAL "2";"4=ptt          5=woon
   6=btwnr  ", INK 1,"7=telnr
   8=ind     9=terug  "
  240 PRINT AT VAL "21",VAL "0";"
D=instring",STRING$(VAL "22",e$)
  250 PROC off
  260 POKE x17,VAL "8"
  270 DO
  280 GET wb: EXIT IF wb>VAL "1"
OR wb<VAL "13"
  290 LOOP
  300 IF wb=9 THEN PROC menu
  310 POKE x17,VAL "0"
  320 PROC keuze
  330 LET xx=x: LET yy=y
  340 IF wb=VAL "10" THEN PROC se
lectie: PROC zoeken
  350 IF wb=VAL "11" THEN PRINT A
T VAL "17",VAL "0";TAB VAL "31";
e$: PROC vantot
```

```
 360 IF wb=VAL "12" THEN PRINT A
T VAL "17",VAL "0",TAB VAL "31";
e$: PROC alfa
 370 IF wb=VAL "13" THEN PROC in
str: PROC zoeken
 380 IF wb<=VAL "8" THEN DO : IN
PUT "te zoeken ";ip$(wb,VAL "3"
TO ));: LINE n$: EXIT IF n$<>"":
LOOP
 390 IF LEN n$<x2 THEN LET n$=n$
+" "
 400 IF n$="s" THEN PROC zoeken
 410 PRINT INVERSE VAL "1",AT VA
L "0",VAL "0";p$(wb);AT VAL "0",
z,n$: PROC test
 420 FOR m=x1 TO j
 430 IF INSTRING(x1,b$(m,x TO y)
,n$( TO x2/))<>VAL "0" THEN LET c
o=co+VAL "1": IF test=VAL "1" TH
EN PROC poke  PROC print2: ELSE
PROC tussen
 440 NEXT m
 450 PROC datatel: PROC niet
 460 END PROC . REM einde zoeken
 480 DEF PROC ingave
 490 PROC on
 500 PRINT INVERSE VAL "0";AT VA
L "20",VAL "0";STRING$(VAL "32",
e$)
 510 PRINT INK VAL "3";AT VAL "1
7",VAL "0";"1-klantnr 2-naam   3
-straat      "; INK VAL "2";"4-pt
tcode 5-woonpl 6-btwnr       "; I
NK VAL "1";"7-tel.nr. 8-indic. 9
-geen mut.  "
 520 PROC off  POKE x17,VAL "0"
 530 PRINT AT VAL "21",VAL "0",S
TRING$(VAL "32",e$)
 540 PRINT AT VAL "20",VAL "9","
ENTER nummer"
 550 GET wb: POKE x17,VAL "0"
 555 IF wb>9 THEN BEEP .1,.1
 560 LET ch=ch+VAL "1"
 570 LET aa=VAL "1": PROC keuze
 580 PRINT BRIGHT VAL "1",AT VAL
 "20",VAL "7","* voor correctie"
 590 IF wb<VAL "9" THEN INPUT ip
$(wb,VAL "3" TO )), LINE a$: IF
a$="*" THEN BEEP VAL "1",VAL "1"
 PROC ingave: ELSE LET b$(m,x T
O y)=a$  PRINT AT (wb+VAL "2"),z
,b$(m,x TO y)
 600 IF wb<>VAL "9" THEN PROC in
gave
 610 LET x=xx: LET y=yy: PROC of
f: PRINT AT VAL "0",VAL "23",STR
ING$(VAL "9",e$): LET aa=VAL "0"
: END PROC
 620 DEF PROC scherm
 630 PROC on: FOR k=VAL "2" TO V
AL "16" STEP VAL "2"
```

```
 640 PRINT AT k,VAL "0",p$(k/VAL
 "2")
 650 NEXT k: PROC off: END PROC
 660 DEF PROC print
 670 PRINT AT VAL "0",VAL "29";m
 680 PRINT AT VAL "2",z;b$(m, TO
 x2);AT x2,z,b$(m,x3 TO x4)
 690 PRINT AT VAL "6",z;b$(m,x5
TO x6);AT VAL "8",z;b$(m,x7 TO x
8),AT z,z,b$(m,x9 TO x10)
 700 PRINT AT VAL "12",z;b$(m,x1
1 TO x12);AT VAL "14",z;b$(m,x13
 TO x14)
 710 PRINT AT VAL "16",z,b$(m,x1
5 TO )
 720 IF INKEY$="s" THEN PROC zoe
ken
 730 END PROC
 740 DEF PROC clear
 750 FOR k=VAL "2" TO VAL "16" S
TEP VAL "2"
 760 PRINT AT k,z-VAL "1";STRING
$(VAL "22",e$)
 770 NEXT k
 780 PRINT AT VAL "1",VAL "0";ST
RING$(VAL "32",e$)
 790 PROC clear2
 800 END PROC
 810 DEF PROC selectie
 820 PROC on
 830 PRINT INK VAL "1";AT VAL "1
7",VAL "0";"1-klantnr. 2-naam
 3-straat   "; INK VAL "2";"4-pt
tcode  5-woonpl. 6-btwnr.   "; I
NK VAL "3";"7-tel.nr.  8-indic.
 9-starten  ", INK VAL "4";"A=va
n tot";TAB VAL "31";e$
 840 PRINT INK VAL "5",AT VAL "2
1",VAL "0";"welke selecties:",ST
RING$(VAL "16",e$)
 850 PROC off: PROC leeg
 860 DO
 870 DO
 880 POKE x17,VAL "8"
 890 GET ke
 900 POKE x17,VAL "0"
 910 EXIT IF ke>VAL "1" OR ke<VA
L "10"
 920 LOOP
 930 IF ke=VAL "1" THEN INPUT (p
$(ke)); LINE c$
 940 IF ke=VAL "2" THEN INPUT (p
$(ke)); LINE d$
 950 IF ke=VAL "3" THEN INPUT (p
$(ke)); LINE f$
 960 IF ke=VAL "4" THEN INPUT (p
$(ke)); LINE g$
 970 IF ke=VAL "5" THEN INPUT (p
$(ke)); LINE h$
 980 IF ke=VAL "6" THEN INPUT (p
$(ke)); LINE i$
```

```
 990 IF ke=VAL "7" THEN INPUT (p
$(ke)); LINE k$
1000 IF ke=VAL "8" THEN INPUT (p
$(ke)); LINE l$
1010 IF ke=VAL "10" THEN PROC va
ntot
1020 EXIT IF ke=VAL "9"
1030 LOOP
1040 PROC clear: PROC test
1050 FOR m=x1 TO j
1060 PROC and
1070 NEXT m
1080 PROC datatel: PROC niet: EN
D PROC
1090 DEF PROC vantot
1100 PRINT #0,AT VAL "1",VAL "0"
; BRIGHT VAL "1","keuze (1-8)";S
TRING$(VAL "21",e$): GET wb: IF
wb<VAL "1" OR wb>VAL "8" THEN PR
OC vantot
1110 PROC keuze: INPUT (p$(wb));
"van "; LINE v$;" tot "; LINE w$
1120 PROC on: PRINT AT VAL "0",V
AL "0";p$(wb,VAL "3" TO );" van
";v$;" tot ";w$
1130 PROC off: PROC clear: PROC
test
1140 FOR m=x1 TO j
1150 IF b$(m,x TO y)>=v$ TO LEN
 v$) AND b$(m,x TO y)<=w$( TO LE
N w$) THEN PROC and
1160 NEXT m
1170 PROC datatel: PROC niet: EN
D PROC
1180 DEF PROC and
1190 IF b$(m,x1 TO VAL "0"+LEN c
$)=c$ THEN IF b$(m,x3 TO VAL "4"
+LEN d$)=d$ AND b$(m,x5 TO VAL "
25"+LEN f$)=f$ AND b$(m,x7 TO VA
L "46"+LEN g$)=g$ THEN IF b$(m,x
9 TO VAL "51"+LEN h$)=h$ AND b$(
m,x11 TO VAL "67"+LEN i$)=i$ THE
N IF b$(m,x13 TO VAL "79"+LEN k$
)=k$ AND b$(m,x15 TO VAL "95"+LE
N l$)=l$ THEN LET co=co+VAL "1".
 IF test=VAL "1" THEN PROC poke:
 PROC print2: ELSE PROC tussen
1200 END PROC
1210 DEF PROC alfa
1220 PRINT AT VAL "21",VAL "0";"
welke rubriek ? (1-8)",STRING$(V
AL "11"," "); GET wb
1230 IF wb<VAL "1" OR wb>VAL "8"
 THEN PROC alfa
1240 PRINT #0,AT VAL "1",VAL "0"
; BRIGHT VAL "1","alfabethisch o
p ",p$(wb,VAL "3" TO )
1250 PROC keuze: PROC sort: PROC
 test
1260 FOR m=x1 TO j
```

```
1270 IF test=VAL "1" THEN LET co
=co+VAL "1": PROC poke: PROC pri
nt2: ELSE PROC print: PROC tusse
n
1280 NEXT m: PRINT #0;AT VAL "1"
,VAL "0",STRING$(VAL "32",e$): P
ROC datatel: PROC niet
1290 END PROC
1300 DEF PROC sort: SORT b$(x1 T
O j)(x TO ): END PROC
1310 DEF PROC clear2: PRINT AT V
AL "0",VAL "20";STRING$(X2,e$):
FOR k=VAL "17" TO VAL "21": PRIN
T AT k,VAL "0";STRING$(VAL "32",
e$): NEXT k: END PROC
1320 DEF PROC verwijder
1330 PRINT FLASH VAL "1",AT VAL
"17",VAL "0";"zeker (j)"; FLASH
VAL "0";STRING$(VAL "23",e$): GE
T z$
1340 IF z$="j" THEN PROC clear:
FOR k=m TO j: LET b$(k)=b$(k+VAL
 "1"): NEXT k: LET j=j-VAL "1":
LET del=del+VAL "1": LET m=m-VAL
 "1"
1350 END PROC
1360 DEF PROC tussen
1370 PROC print: PROC on: PRINT
AT VAL "21",VAL "0";"m=mutatie
s=stop  v=verwijder ": PROC off
1380 GET x$
1390 IF x$="v" THEN PROC verwijd
er
1400 IF x$="m" THEN PROC ingave:
 PROC clear
1410 IF x$="s" THEN LET test=VAL
 "0"  PROC datatel: PROC clear:
PRINT #0;AT VAL "1",VAL "0";STRI
NG$(VAL "32",e$): PROC zoeken
1420 END PROC
1430 DEF PROC instr
1440 PROC clear: INPUT BRIGHT VA
L "1";"geef INSTRING "; LINE a$
1450 PROC on: PRINT AT VAL "0",V
AL "0";"INSTRING "; INVERSE VAL
"0";AT VAL "0",z,a$: PROC off
1460 PROC test  FOR m=x1 TO j
1470 IF INSTRING(x1,b$(m, TO ),a
$)<>VAL "0" THEN LET co=co+VAL "
1": PROC poke: IF test=VAL "1" T
HEN PROC print2: ELSE PROC tusse
n
1480 NEXT m
1490 PROC datatel: PROC niet
1500 END PROC
```

```
1510 DEF PROC inputvar: DIM p$(1
1,9): LET p$(1)="1-KL NR": LET P
$(2)="2-NAAM": LET P$(3)="3-STRA
AT": LET P$(4)="4-PTT NR": LET P
$(5)="5-WOONPL": LET P$(6)="6-BT
W NR": LET P$(7)="7-TEL NR": LET
 P$(8)="8-INDIC": LET p$(9)=" FO
UND": LET p$(10)=" READ": LET p$
(11)=" DELETE ": END PROC
1520 DEF PROC datatel
1530 IF test=VAL "1" AND kk<>VAL
 "0" THEN PROC keuze2
1540 CLS
1550 PROC on: INK VAL "1": PRINT
 AT VAL "17",VAL "0","DATA",p$(V
AL "9"), USING "###";co: INK VAL
 "2","DATA";p$(VAL "10"), USING
"###";m: INK VAL "3","DATA ",p$(
VAL "11", TO VAL "2"); USING "##
#";del: INK VAL "4";"DATA CHANGE
D ", USING "###",ch: INK VAL "5"
;"ACKNOWLEDGE ", FLASH VAL "1
";"?"; FLASH VAL "0",STRING$(VAL
 "16",e$): PROC off: PAUSE VAL "
0": END PROC
1560 DEF PROC print2
1570 PRINT AT kk,VAL "0";m,AT kk
,VAL "3";b$(m,x3 TO VAL "20")
1580 IF wb=VAL "2" OR wb=VAL "5"
 THEN PRINT AT kk,VAL "20",b$(m,
x9 TO VAL "64")
1590 IF wb=VAL "1" THEN PRINT AT
 kk,VAL "20";b$(m, TO x2)
1600 IF wb=VAL "4" THEN PRINT AT
 kk,VAL "20";b$(m,x7 TO x8)
1610 IF wb=VAL "3" THEN PRINT AT
 kk,VAL "19":b$(m,x5 TO VAL "38"
)
1620 IF wb=VAL "7" THEN PRINT AT
 kk,VAL "20";b$(m,x13 TO VAL "90
")
1630 IF wb>VAL "7" THEN PRINT AT
 kk,VAL "20";b$(m,x9 TO VAL "62"
)
1640 IF INKEY$="s" THEN CLS : PR
OC zoeken
1650 LET kk=kk+VAL "1": IF kk=VA
L "21" THEN PROC keuze2: CLS : L
ET kk=VAL "0": ELSE IF m=j AND k
k>VAL "0" THEN PROC keuze2: PROC
 keuze2
1660 END PROC
1670 DEF PROC keuze2
1680 PROC on: PRINT INK VAL "1";
AT VAL "21",VAL "0";"ENTER=verde
r"; INK VAL "2","*=terug"; INK V
AL "3";"ENTER nummer": PROC off
1690 INPUT LINE o$
1700 IF o$="" THEN LET angle=ang
le+VAL "1": LET kk=VAL "0": GO T
O 1730
```

```
1710 IF o$="*" THEN CLS : PROC z
oeken
1720 LET cijf=VAL "0": PROC cijf
er: IF cijf=VAL "1" THEN PROC ke
uze2
1730 END PROC
1740 DEF PROC cijfer
1750 LET cijf=cijf+VAL "1"
1760 FOR v=VAL "1" TO LEN o$
1770 IF CODE o$(v)<VAL "48" OR C
ODE o$(v)>VAL "57" THEN GO TO 18
00: ELSE NEXT v: LET cijf=cijf+V
AL "1"
1780 IF VAL o$<=VAL "0" OR VAL o
$>j THEN PROC keuze2
1790 LET km=wb: LET kl=m: LET m=
VAL o$: CLS : PROC scherm: PROC
tussen : LET m=kl: LET wb=km: CLS
 : PROC pk
1800 END PROC
1810 DEF PROC pk
1820 LET kk=VAL "0": IF angle TH
EN LET pke=pke+VAL "21"
1830 FOR w=pke TO pk-VAL "1"
1840 LET p=PEEK w
1850 IF NOT p THEN PAUSE VAL "0"
: PROC zoeken: ELSE LET m=p: PRO
C print2
1860 NEXT w
1870 IF NOT angle THEN PROC keuz
e2
1880 END PROC
1890 DEF PROC saveload
1900 CLS
1910 PRINT ''"1 - Save programma
"''"2 - Save data klanten"''"3 -
 Load data klanten"''"4 - Terug
naar hoofdmenu"
1920 GET a
1930 IF a=VAL "1" THEN PROC save
BB
1940 IF a=VAL "2" THEN PROC save
 data
1950 IF a=VAL "3" THEN PROC load
 data
1960 PROC menu: END PROC
1970 DEF PROC load data
1980 CLS   DO
1990 INPUT "naam "; LINE a$
2000 EXIT IF LEN a$<=10 AND LEN
a$>0
2010 LOOP
2020 LET k=VAL "1"
2030 OPEN #x2;"m";x1;a$
2040 DIM b$(VAL "150",x16)
2050 DO WHILE NOT EOF(x2)
2060 INPUT #x2;b$(k)
2070 LET k=k+x1
2080 LOOP
2090 CLOSE #x2: LET j=k
2100 END PROC
```

```
2110 DEF PROC save data
2120 CLS : DO
2130 INPUT "naam "; LINE a$
2140 EXIT IF LEN a$<=10 AND LEN
a$>0
2150 LOOP
2160 OPEN #x2;"m";x1;a$
2170 FOR k=x1 TO j
2180 PRINT #x2;b$(k)
2190 NEXT k
2200 CLOSE #x2
2210 END PROC
2220 DEF PROC save66
2230 CLEAR : LET rt=DPEEK(VAL "2
3730"): RANDOMIZE USR VAL "59904
": SAVE *"m";1;"run" LINE 2240.
SAVE *"m";1;"66"CODE rt+VAL "1",
VAL "65367"-rt: RANDOMIZE USR VA
L "58419": PROC menu
2240 CLEAR rt: LOAD *"m";1;"66"C
ODE : CLS : RANDOMIZE USR VAL "5
8419": PROC menu
2250 END PROC
2260 DEF PROC varia: LET m=1: LE
T pk=VAL "65300": LET ch=VAL "0"
: LET kk=ch: LET del=ch: LET co=
ch: LET x=1: LET xx=x: LET y=VAL
 "4": LET yy=y: LET z=VAL "10":
LET a$=" ": LET aa=ch: LET pke=p
k: LET angle=ch: LET test=ch: EN
D PROC
2270 DEF PROC keuze
2280 IF wb<VAL "9" AND aa=VAL "0
" THEN PROC clear
2290 IF wb=VAL "1" THEN LET x=VA
L "1": LET y=VAL "4"
2300 IF wb=VAL "2" THEN LET x=x3
: IF aa=VAL "1" THEN LET y=x4: E
LSE LET y=VAL "8"
2310 IF wb=VAL "3" THEN LET x=x5
: IF aa=VAL "1" THEN LET y=x6: E
LSE LET y=VAL "29"
2320 IF wb=x2 THEN LET x=x7: IF
aa=VAL "1" THEN LET y=x8: ELSE L
ET y=VAL "50"
2330 IF wb=x3 THEN LET x=x9: IF
aa=VAL "1" THEN LET y=x10: ELSE
LET y=VAL "55"
2340 IF wb=VAL "6" THEN LET x=x1
1: IF aa=VAL "1" THEN LET y=x12.
 ELSE LET y=VAL "71"
2350 IF wb=VAL "7" THEN LET x=x1
3: IF aa=VAL "1" THEN LET y=x14:
 ELSE LET y=VAL "83"
2360 IF wb=VAL "8" THEN LET x=x1
5: IF aa=VAL "1" THEN LET y=x16:
 ELSE LET y=VAL "99"
2370 END PROC : REM einde keuze
```

```
2380 DEF PROC poke: POKE pk,m: L
ET pk=pk+VAL "1": IF pk=VAL "553
99" THEN BEEP VAL "1",VAL "1": P
AUSE VAL "0": CLS : PROC zoeken:
 ELSE END PROC
2390 DEF PROC test: PRINT AT VAL
 "21",VAL "0","1(overzicht) of 2
(apart)  ": GET test: IF test=V
AL "1" THEN CLS
2400 IF test>VAL "2" OR test<VAL
 "1" THEN PROC test
2410 END PROC
2420 DEF PROC leeg: LET c$="": L
ET d$="": LET f$="": LET g$="":
LET h$="": LET i$="": LET k$="":
 LET l$="": END PROC
2430 DEF PROC off: BRIGHT VAL "0
": INVERSE VAL "0": INK VAL "0":
 END PROC
2440 DEF PROC on: BRIGHT VAL "1"
: INVERSE VAL "1": END PROC
2450 DEF PROC niet: IF test=VAL
"1" THEN CLS
2460 PROC clear: PROC zoeken
2470 END PROC
2480 DEF PROC menu
2490 PROC varia
2500 PROC inputvar: LET x1=VAL "
1": LET x2=VAL "4": LET x3=VAL "
5": LET x4=VAL "25": LET x5=VAL
"26": LET x6=VAL "46": LET x7=VA
L "47": LET x8=VAL "51": LET x9=
VAL "52": LET x10=VAL "67": LET
x11=VAL "68": LET x12=VAL "79"
LET x13=VAL "80": LET x14=VAL "9
5": LET x15=VAL "96": LET x16=VA
L "100": LET x17=VAL "23658"
2510 REM DELETE 1510 TO 1510: DE
LETE 2500 TO 2510
2520 PROC off: CLS
2530 PRINT '" 1 - invoer van DA
TA "'" 2 - zoeken van DATA "'"
 3 - save/load  DATA "'" 4 - st
op"
2540 PRINT #0; INVERSE 1; INK VA
L "3";AT VAL "1",VAL "1":"ENTER
uw keuze": GET hmenukeus
2550 IF hmenukeus=1 THEN PROC in
voer
2560 IF hmenukeus=2 THEN PROC zo
eken
2570 IF hmenukeus=3 THEN PROC sa
veload
2580 IF hmenukeus=4 THEN STOP
2590 END PROC
```

```
1970 DEF PROC load data
1980 CLS : DO
1990 INPUT "naam "; LINE a$
2000 EXIT IF LEN a$<=10 AND LEN
a$>0
2010 LOOP
2020 OPEN #x2,"m";x1;a$
2030 DIM b$(VAL "150",x16)
2040 FOR k=x1 TO VAL "150"
2050 INPUT #x2,b$(k)
2060 IF b$(k, TO VAL "3")="EOF"
THEN GO TO VAL "2080"
2070 NEXT k
2080 CLOSE #x2: LET j=k: PROC me
nu
2090 END PROC
2100 DEF PROC save data
2110 CLS : DO
2120 INPUT "naam "; LINE a$
2130 EXIT IF LEN a$<=10 AND LEN
a$>0
2140 LOOP
2150 OPEN #x2,"m",x1;a$
2160 FOR k=x1 TO j
2170 PRINT #x2;b$(k)
2180 NEXT k
2190 LET b$(j, TO VAL "3")="EOF"
: PRINT #x2,b$(j)
2200 CLOSE #x2: PROC menu
2210 END PROC
```

```
1970 DEF PROC load data
1980 CLS : DO
1990 INPUT "naam "; LINE a$
2000 EXIT IF LEN a$<=10 AND LEN
a$>0
2010 LOOP
2020 DIM b$(VAL "150",x16)
2030 LOAD a$ DATA b$()
2040 FOR k=x1 TO VAL "150"
2050 IF b$(k, TO VAL "3")="EOF"
THEN GO TO 2070
2060 NEXT k
2070 LET j=k: PROC menu
2080 END PROC
2090 DEF PROC save data
2100 CLS : DO
2110 INPUT "naam "; LINE a$
2120 EXIT IF LEN a$<=10 AND LEN
a$>0
2130 LOOP
2140 IF j<VAL "150" THEN LET b$(
j+VAL "1", TO VAL "3")="EOF"
2150 SAVE a$ DATA b$()
2160 PROC menu
2170 END PROC
2180 DEF PROC saveBB
2190 CLEAR : LET rt=DPEEK(VAL "2
3730") RANDOMIZE USR VAL "59904
": SAVE "naam" LINE 2200: POKE D
PEEK(23631)+2,151: SAVE "BB"CODE
 rt+VAL "1",VAL "65367"-rt: RAND
OMIZE USR VAL "58419": PROC menu
2200 CLEAR rt: LOAD "BB"CODE : C
LS : RANDOMIZE USR VAL "58419":
PROC menu
2210 END PROC
```

6.10 Programma Conversies.

Het conversieprogramma is vooral bedoeld om de meer ingewikkelde syntax aan te tonen die ontstaat wannear meerdere conversies worden uitgevoerd na elkaar.

De procedure conv 1 - waarbij uitgegaan wordt van een binair getal - is afwezig. De enige mogelijke directe conversie hiervoor is: PRINT BIN 1010010, waarbij geen variabelen kunnen ingebracht.

Conversie 2 gaat uit van een 2-karakterstring, conversie 3 van een decimaal getal en conversie 4 van een hexadecimale, terwijl conversie 5 - evenals nr 3 - uitgaat van een gewoon decimaal getel.

Indien het programma u bevalt, kan u door JOIN de PRINT-instructies per conversie aan elkaar lijmen en daarna de PRINT's verwijderen.

Na het runnen wordt er gevraagd welke conversie u wilt. U drukt dan het cijfer dat wijst naar het stelsel waarin uw vertrekgetal is geschreven.

Aangezien er geen enkele controle wordt uitgevoerd op de INPUT van uw aanvangsgetal in regel 180, werd op regel 5 een ON ERROR gebruikt.

CONVERSIES

```
  5 ON ERROR 20
 20 PROC CONV MENU
 30 PAUSE 0: GO TO 20
 40 STOP
 50 REM
 60 LET T$="CONVERSIES"
 70 DIM R$(5,5)
 80 LET R$(1)="BIN$()"
 90 LET R$(2)="CHAR$()"
100 LET R$(3)="DEC()"
110 LET R$(4)="HEX$()"
120 LET R$(5)="NUMBER()"
130 REM
140 DEF PROC CONV MENU: CLS
150 PRINT ''TAB 4;T$'': FOR F=1
 TO 5: PRINT USING "00";F;" ";R$
(F);TAB 12;" TO ": NEXT F: PRINT
 '''' 7 TO STOP ";#0;AT 0,0;" HAA
L EEN GETAL "
160 GET HH: IF HH>7 THEN GO TO
160
170 IF HH=7 THEN STOP
180 INPUT \"GEEF "+R$(HH)); LIN
E I$: PRINT AT 2,18;I$
190 IF HH=4 THEN PROC CONV4
200 IF HH=5 THEN PROC CONV3
210 IF HH=2 THEN PROC CONV2:
    ELSE IF HH=1 OR HH=3 THEN P
ROC CONV3
220 PRINT #0;AT 0,0;"ENTER om v
erder te gaan"
230 END PROC
240 DEF PROC CONV3  REM DEC
250 PRINT AT 4,15;BIN$(VAL I$)
260 PRINT AT 5,16;CHAR$(VAL I$)
270 PRINT AT 6,16;;I$
280 PRINT AT 7,16;HEX$(VAL I$)
290 PRINT AT 8,16;I$: END PROC
300 DEF PROC CONV4  REM HEX
310 PRINT AT 4,16,BIN$(DEC(I$))
320 PRINT AT 5,16;CHAR$(DEC(I$)
)
330 PRINT AT 6,16;DEC(I$)
340 PRINT AT 7,16;I$
350 PRINT AT 8,16;NUMBER(CHAR$(
DEC(I$))): END PROC
360 DEF PROC CONV2  REM CHAR$
370 PRINT AT 4,16;BIN$(NUMBER(I
$))
380 PRINT AT 5,16;I$
390 PRINT AT 6,16;NUMBER(I$)
400 PRINT AT 7,16;HEX$(NUMBER(I
$))
410 PRINT AT 8,16;NUMBER(I$): E
ND PROC
```

6.11 Programma Bestandaprogramma.

Onderstaend programme is een veriatie op het adressen programma. het verschil is dat met dit programme (in deze vorm tenminste) de DATA niet geselecteerd of gesorteerd kunnen worden.
Het voordeel is echter det elle DATA weggePOKEd wordt zonder spaties mee te nemen, waardoor het eentel DATA wallicht twee a drie maal zo groot is. De DATA wordt als volgt weggePOKEd:

1 positie om het begin van een "string" aan te duiden (keyword COPY),
2 posities om de LEN van de "string" aan te duidan en
x posities voor de "string" zelf.

Het programma in zijn huidige vorm vraagt bij het initialiseren om vier rubriekennaman die vervolgens gePOKEt (bijvoorbeeld: naam, adres, woonplaats en telefoon).
Deze gegevens worden bij de DATA gevoegd en meegeSAVEd .

N$, A$, W$ an T$ zijn de inputstrings. Zodre deze goed zijn bevonden worden deze opgenomen in Z$ (LET Z$= N$ + "#" + A$ + "#" + W$ + "#" + T$). Het " # " teken wordt hierbij gebruikt om de subrecords aan te duiden.
Het begin teken, de LEN en de string zelf worden weggePOKEd en het eindadres wordt aangepast.

Bij het terugzoeken wordt gezocht in het gebied venaf het eerst gebruikte adres tot een het laast gebruikte adres. Indien gevonden, wordt het tijdelijk opgeslagen in A$, waarna PROC(edure) controle wordt aangeroepen. Deze procedure zoekt terug in het geheugen neer het begin en de LEN van de oorspronkelijke "string". Zodra deze is gevonden wordt A$ gelijk gemaakt aan de oorspronkelijk weggePOKEte string.

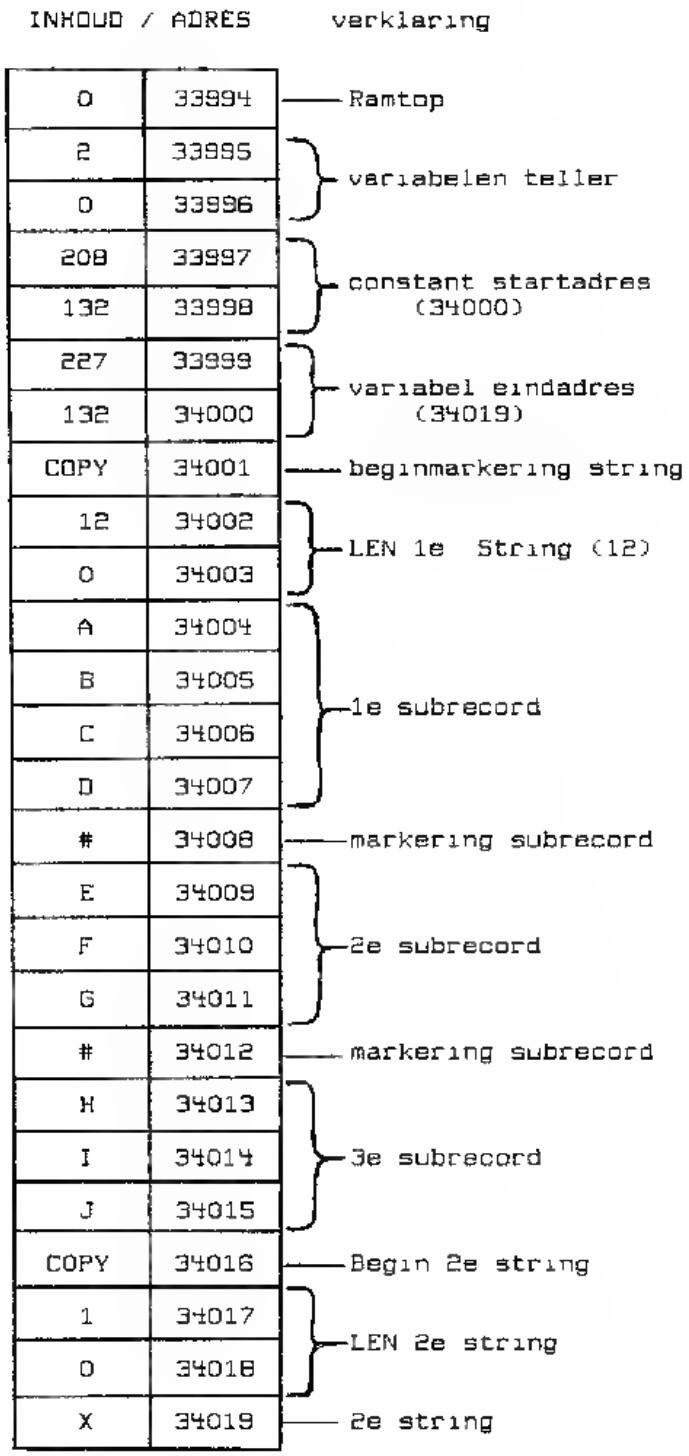
De PRINT-routine zorgt ervoor dat het " # " teken vervangen wordt door een komme en det er op een nieuwe regel verder gePRINT wordt. Na het PRINTen kunt U terug naar het menu gaen, een volgend edres op zoeken of het gevonden adres wijzigen.
Het wijzigen heeft tot gevolg dat het scherm eerst geSCREEN$d moet worden, waarbij de oorspronkelijke strings wederom gecreeerd wordan.

U kunt nu in een DO-LOOP keuze meken door de te wijzigen rubrieken aan te geven en in te vullen. Keuze 5 heeft tot gevolg det er niets gewijzigd wordt en keuze 6 heeft tot gevolg det de wijzigingen verwerkt worden.
Uw wijzigingen kunnen tot gevolg hebben det er tussen de LEN's van de oorspronkelijke en de nieuwe string positieve en negatieve verschillen kunnen ontstaan. Stel U heeft 10 records ingebracht en u wenst nu bijvoorbeeld record twee wijzigen. Record 1 blijft staan waer het stond, records 3 tot en met 10 worden "opgeschoven" (afhenkelijk van het verschil ven de LEN's naer boven of naer onderen) en vervolgens wordt de nieuwe string er tussen gePOKEt. Uitaraard worden hierbij alle LEN adrassen aangepest evenals het eindadres.

Het programma is zeer eenvoudig uit te breiden en aan te passen aan Uw eigen wensen. Indien gewenst kunt U het eental rubrieken verhogen. U dient dan enkele routines overeenkomstig uit te breiden / aan te passen.
Het programme meg geRUNd of geCLEARd worden; de DATA wordt enkel verwijdert als U de stekker uit het stopkontakt trekt (of RANDOMIZE USR 0 gaaft.)

PRINT MEMORY$ () (34000 TO DPEEK (33999)) geeft U zicht op de manier hoe alles wordt opgeslegen.

6.11 INDELING VAN GEHEUGEN BIJ HET POKEn VAN STRINGS



```
  10 REM 33954 TOT 33993 RUBRIEK
GEGEVENS 33995 GEEFT TELLER VANA
ANTAL DATA INGEVOERD 33997 GEEFT
 STARTADRES 33999 GEEFT EIND ADR
ES
  20 PROC MENU
  30 DEF PROC INITIALISEREN
  40 CLS
  50 POKE  33954,STRING$(1E3," "
)
  60 POKE  23658,8
  70 DPOKE 33995,1
  80 DPOKE 33997,34E3
  90 DPOKE 33999,34001
 100 LET ADR=33954
 110 DIM B$(4,10)
 120 FOR F=1 TO 4
 130 INPUT "RUBRIEK ";(F)," "; L
INE B$(F)
 140 POKE ADR,B$(F)
 150 PRINT f;" ";MEMORY$()(ADR T
O ADR+9)
 160 LET ADR=ADR+10
 170 NEXT F
 180 END PROC
 190 DEF PROC INBRENGEN
 200 CLS
 210 PRINT AT 10,0,"INITIALISERE
N ? 1=JA / ENTER=NEE"; GET INIT:
 IF INIT=1 THEN PROC INITIALISER
EN: CLEAR
 220 LET ADRES=DPEEK(33999)
 230 LET ADR=33954
 240 DO WHILE ADRES<55E3
 250 CLS
 260 PRINT AT 0,20;DPEEK(33995)
 270 PRINT AT 18,0;" STOP VOOR M
ENU"
 280 POKE 23658,8
 290 INPUT (MEMORY$()(ADR TO ADR
+9)+CHR$ 13); LINE N$
 300 PRINT AT 18,0;STRING$(20,"
")
 310 EXIT IF N$=" STOP "
 320 PRINT AT 0,0;N$
 330 INPUT (MEMORY$()(ADR+10 TO
ADR+19)+CHR$ 13), LINE A$: PRINT
 A$
 340 INPUT (MEMORY$()(ADR+20 TO
ADR+29)+CHR$ 13); LINE W$: PRINT
 W$
 350 INPUT (MEMORY$()(ADR+30 TO
ADR+39)+CHR$ 13); LINE T$: PRINT
 T$
 360 PRINT AT 21,0;"GOED ? 0=NEE
, REST=JA"
 370 GET GOED
 380 IF NOT GOED THEN GO TO 250
 390 LET Z$=N$+"#"+A$+"#"+W$+"#"
+T$
```

```
 400 POKE ADRES," COPY "
 410 DPOKE ADRES+1,LEN Z$
 420 POKE ADRES+3,Z$
 430 LET ADRES=ADRES+(LEN Z$)+3
 440 DPOKE 33995,(DPEEK(33995))+
1
 450 LOOP
 460 DPOKE 33999,ADRES: PROC MEN
U
 470 END PROC
 480 DEF PROC ZOEKEN
 490 CLS
 500 LET ADRES=DPEEK(33997)
 510 PRINT AT 21,0;" STOP  OM TE
 STOPPEN"
 520 POKE 23658,8
 530 INPUT "TE ZOEKEN "; LINE G$
 540 IF G$=" STOP " THEN PROC ME
NU
 550 LET LEN=LEN G$
 560 LET TEL=0: LET TELLER2=1
 570 DO
 580 LET BEGIN=INSTRING(ADRES,ME
MORY$()( TO DPEEK(33999)),G$)
 590 EXIT IF NOT BEGIN
 600 LET A$=MEMORY$()(BEGIN TO B
EGIN-1+LEN)
 610 PROC CONTROLE
 620 LET BEGIN=BEGIN+1
 630 LET ADRES=BEGIN
 640 LOOP
 650 IF TEL>1 THEN PRINT AT 21,0
;"KONT NIET MEER VOOR ";STRING$
(5," "): PAUSE 0: PROC ZOEKEN: E
LSE PRINT AT 21,0;"KONT NIET VOO
R",STRING$(10," "): PAUSE 0: PRO
C ZOEKEN
 660 DEF PROC CONTROLE
 670 REM BEGIN (G) ZOEKEN
 680 LET CONTROL=BEGIN
 690 DO
 700 EXIT IF CHR$ PEEK CONTROL="
 COPY "
 710 LET CONTROL=CONTROL-1

 720 LOOP
 730 REM PRINTEN
 740 LET A$=MEMORY$()(CONTROL+3
TO (CONTROL+3)+DPEEK(CONTROL+1))
 750 LET Y=0: LET Z=0
 760 FOR G=1 TO LEN A$-1
 770 REM # NIEUWE REGEL PRINTEN
 780 REM # VERVANGEN DOOR ,
 790 IF INSTRING(1,A$(G),"#") TH
EN LET A$(G)=",": PRINT ",";STRI
NG$(5," "): LET Y=Y+1: LET Z=-1
 800 PRINT AT Y,Z;A$(G);
 810 LET TEL=TEL+1
 820 LET Z=Z+1: IF Z=31 THEN LET
 Y=Y+1: LET Z=0
 830 NEXT G
```

```
 840 PRINT ".";STRING$(5," ")
 850 PRINT AT 0,29;TELLER2
 860 LET TELLER2=TELLER2+1
 870 BEEP .1,.1
 880 PRINT AT 17,0;"ENTER  5  VO
OR MENU"''"ENTER  VOOR VOLGENDE"
''"ENTER  1  OM TE WIJZIGEN": GE
T VOLGENDE
 890 IF VOLGENDE=214 THEN CLS
 900 IF VOLGENDE=1 THEN PROC WIJ
ZIGEN
 910 IF VOLGENDE=28 THEN PROC ME
NU
 920 END PROC
 930 DEF PROC WIJZIGEN
 940 PRINT AT 17,0;"MOMENTJE A.U
.B.";STRING$(145," ")
 950 REM REGELS OPHALEN
 960 LET F=0: LET R=0: LET ADR=3
3954
 970 LET N$="": PROC REGEL
 980 LET A$="": PROC REGEL
 990 LET W$="": PROC REGEL
1000 LET T$="": PROC REGEL
1010 PRINT AT 12,0;"1 ";MEMORY$(
TO ADR+9)''"2 ";MEMORY$(TO ADR+
ADR+10 TO ADR+19)''"3 ";MEMORY$(
ADR+20 TO ADR+29)''"4 ";MEMORY$(
(ADR+30 TO ADR+39)
1020 PRINT "5 GEEN MUTATIE"''"6 M
UTATIES VERWERKEN"
1030 LET ADR=33954
1040 DO
1050 GET WIJZ
1060 POKE 23658,8
1070 IF WIJZ=1 THEN INPUT (MEMOR
Y$)(ADR TO ADR+9),CHR$ 13); LIN
E N$: PRINT AT 0,0;N$;STRING$(29
-LEN N$," ")
1080 IF WIJZ=2 THEN INPUT (MEMOR
Y$)(ADR+10 TO ADR+19)+CHR$ 13);
 LINE A$: PRINT AT 1,0;A$;STRING
$(32-LEN A$," ")

1090 IF WIJZ=3 THEN INPUT (MEMOR
Y$)(ADR+20 TO ADR+29)+CHR$ 13),
 LINE W$: PRINT AT 2,0;W$;STRING
$(32-LEN W$," ")
1100 IF WIJZ=4 THEN INPUT (MEMOR
Y$)(ADR+30 TO ADR+39)+CHR$ 13);
 LINE T$: PRINT AT 3,0;T$;STRING
$(32-LEN T$," ")
1110 EXIT IF WIJZ=5 OR WIJZ=6
1120 LOOP
1130 IF WIJZ=5 THEN CLS : GO TO
1260
1140 LET Z$=N$+"#"+A$+"#"+W$+"#"
+T$
```

```
1150 LET NWLEN=LEN Z$
1160 LET OUDLEN=DPEEK(CONTROL+1)
1170 REM GEWIJZIGDE GEGEVENS WEG
POKEN
1180 LET B$=""
1190 IF OUDLEN>NWLEN THEN LET B$
=STRING$(!OUDLEN-NWLEN)," ")
1200 POKE CONTROL+3+NWLEN,MEMORY
$()((!CONTROL+3)+DPEEK(CONTROL+1
)) TO DPEEK(33999))+B$+"    "
1210 DPOKE 33998,DPEEK(33999)+(N
WLEN-OUDLEN)+LEN B$
1220 DPOKE CONTROL+1,NWLEN
1230 POKE CONTROL+3,Z$
1240 REM RESTANT  OPGESCHOVEN WE
GPOKEN
1250 CLS
1260 END PROC
1270 DEF PROC REGEL
1280 REM RECONSTRUEREN ORIGINELE
 STRINGS
1290 DO
1300>EXIT IF SCRN$(R,F)="," OR S
CRN$(R,F)="."
1310 IF R=0 THEN LET N$=N$+SCRN$
(R,F)
1320 IF R=1 THEN LET A$=A$+SCRN$
(R,F)
1330 IF R=2 THEN LET W$=W$+SCRN$
(R,F)
1340 IF R=3 THEN LET T$=T$+SCRN$
(R,F)
1350 LET F=F+1
1360 LOOP
1370 LET F=0
1380 LET R=R+1
1390 END PROC
1400 DEF PROC ALLESPRINTEN
1410 CLS
1420 LET TEL=0: LET TELLER2=1
1430 LET BEGIN=34004
1440 LET EIND=DPEEK(34002)
1450 DO

1460>EXIT IF BEGIN+EIND>DPEEK(33
999)
1470 LET A$=MEMORY$()(BEGIN TO B
EGIN+EIND)
1480 PROC CONTROLE
1490 LET BEGIN=BEGIN+EIND+3
1500 LET EIND=DPEEK(BEGIN-2)
1510 LOOP
1520 PROC MENU
1530 END PROC
1540 DEF PROC MENU
1550 CLS
1560 PRINT AT 2,0;,,,,"1 INBRENG
EN",,,,"2 ZOEKEN",,,"3 ALLES PR
INTEN",,,,"4 SAVEN EN LOADEN",,,
"5 STOPPEN"
1570 DO
1580 GET KEUZE
1590 IF KEUZE=1 THEN PROC INBREN
GEN
```

```
1600 IF KEUZE=2 THEN PROC ZOEKEN
1610 IF KEUZE=3 THEN PROC ALLESP
RINTEN
1620 IF KEUZE=4 THEN PROC SAVELO
AD
1630 IF KEUZE=5 THEN STOP
1640 LOOP
1650 END PROC
1660 DEF PROC SAVELOAD
1670 CLS
1680 PRINT "1 SAVE PROGRAMMA";,,
,"2 SAVE DATA";,,,,"3 LOAD DATA"
;,,,,"4 HOOFDMENU"
1690 GET SAVE
1700 IF SAVE=1 THEN INPUT "NAAM
"; LINE O$: SAVE *"M";1,O$ LINE
1760: VERIFY *"M";1;O$
1710 IF SAVE=2 THEN INPUT "NAAM
"; LINE O$  SAVE *"m";1;O$CODE 3
3953,DPEEK(33999)-33953: VERIFY
*"m",1;O$CODE 33953,DPEEK(33999)
-33953
1720 IF SAVE=3 THEN INPUT "NAAM
"; LINE O$: LOAD *"m",1,O$CODE
1730 IF SAVE=4 THEN PROC MENU
1740 PROC MENU
1750 END PROC
1760 CLEAR 55400
1770 LOAD *"M";1;"BB"CODE
1780 RANDOMIZE USR 58419
1800 CLEAR 33953: PROC MENU
```

6.12 PRIEMGETALLEN

Priemgetallen zijn getallen waarvan we weten dat deze in eerste instantie alleen deelbaar zijn door zichzelf.
De volgende vier programmaatjes tonen aan op welke verschillende manieren een probleem kan worden opgelost.
Met CLOCK "" en de funktie TIME$() worden de verschillende snelheden van de programmaatjes bepaeld.

```
 10 CLOCK "": CLS
 20 FOR G=3 TO 100 STEP 2: LET P=3
 30 FOR F=2 TO SQR G
 40 IF NOT MOG(G,F) THEN LET P=0
 50 NEXT F
 60 IF P THEN PRINT USING "000";G;" ";
 70 NEXT G
 80 PRINT TIME$()
```

```
 10 DEF PROC SQR
 20 FOR M=3 TO SQR (PRIEM) STEP 2
 30 IF NOT MOD(PRIEM,M) THEN LET PRIEM=PRIEM+2:
 PROC PRIEM
 40 NEXT M
 50 END PROC
 60 DEF PROC PRIEM
 70 DO
 80 PROC SQR
 90 PRINT USING "000";PRIEM;" ";
100 IF PRIEM>=101 THEN POP REGEL:GO TO REGEL+80
110 LET PRIEM=PRIEM+2
120 LOOP
130 END PROC
140 CLS : CLOCK "": LET PRIEM=3: PROC PRIEM
150 STOP
```

```
 10 CLOCK ""
 20 LET PRIEM=1
 30 FOR A=3 TO 100 AND PRIEM<=100 STEP 2
 40 LET PRIEM=PRIEM+2
 50 FOR F=3 TO SQR PRIEM STEP 2
 60 IF NOT MOD(PRIEM,F) THEN NEXT A
 70 NEXT F
 80 PRINT USING "000";PRIEM;" ";
 90 NEXT A
100 PRINT TIME$()
```

```
 10 CLOCK "": CLS
 20 LET PR=49: DIM F(PR)
 30 FOR I=1 TO PR: LET F(I)=I: NEXT I
 40 FOR I=1 TO PR
 50 IF NOT F(I) THEN NEXT I
 60 LET PRM=I+I+1: LET K=I+PRM
 70 DO
 80 EXIT IF K>PR
 90 LET F(K)=0: LET K=K+PRM
100 LOOP
110 PRINT USING "000";PRM;" ";
120 NEXT I
130 PRINT TIME$()
```

CURSOR CONTROLE CODES

De karaktars 8 tot en met 11 hebben geen keyword, maar zijn cursortoetsen, welka in Spectrum-Basic niet effectief kunnen gePRINT worden.

```
CHR$  8  (cursor links)
CHR$  9  (cursor rechts)
CHR$ 10  (cursor onder)
CHR$ 11  (cursor opwaarts)
```

Beta-Basic maakt het PRINTen ervan mogelijk, alsook het gebruik erven in PLOT-instrukties voor strings.

```
20 PRINT " CHR$ 8 (cursor links)" + CHR$ 10 +
         " CHR$ 9 (cursor rechts)"
```

Het karakter 8 of de linkse cursor heeft in Spectrum-Basic een bug: het is niet mogelijk terug te spatieren tot de hoogste lijn van de display uitgaande ven een lager gelegen lijn, en uitgaande ven de hoogste lijn kon terug gespatieerd worden tot de cursor van hat scherm was, zoals in

```
10  PRINT "12" +CHR$ 8+CHR$ 8+CHR$ 8+ "3456"
```

Hierbij wordt bovenaan het scharm '456' gePRINT en verschijnt er een gekleurd blokje rechts van het scherm. Beta-basic corrigeert deze bug.

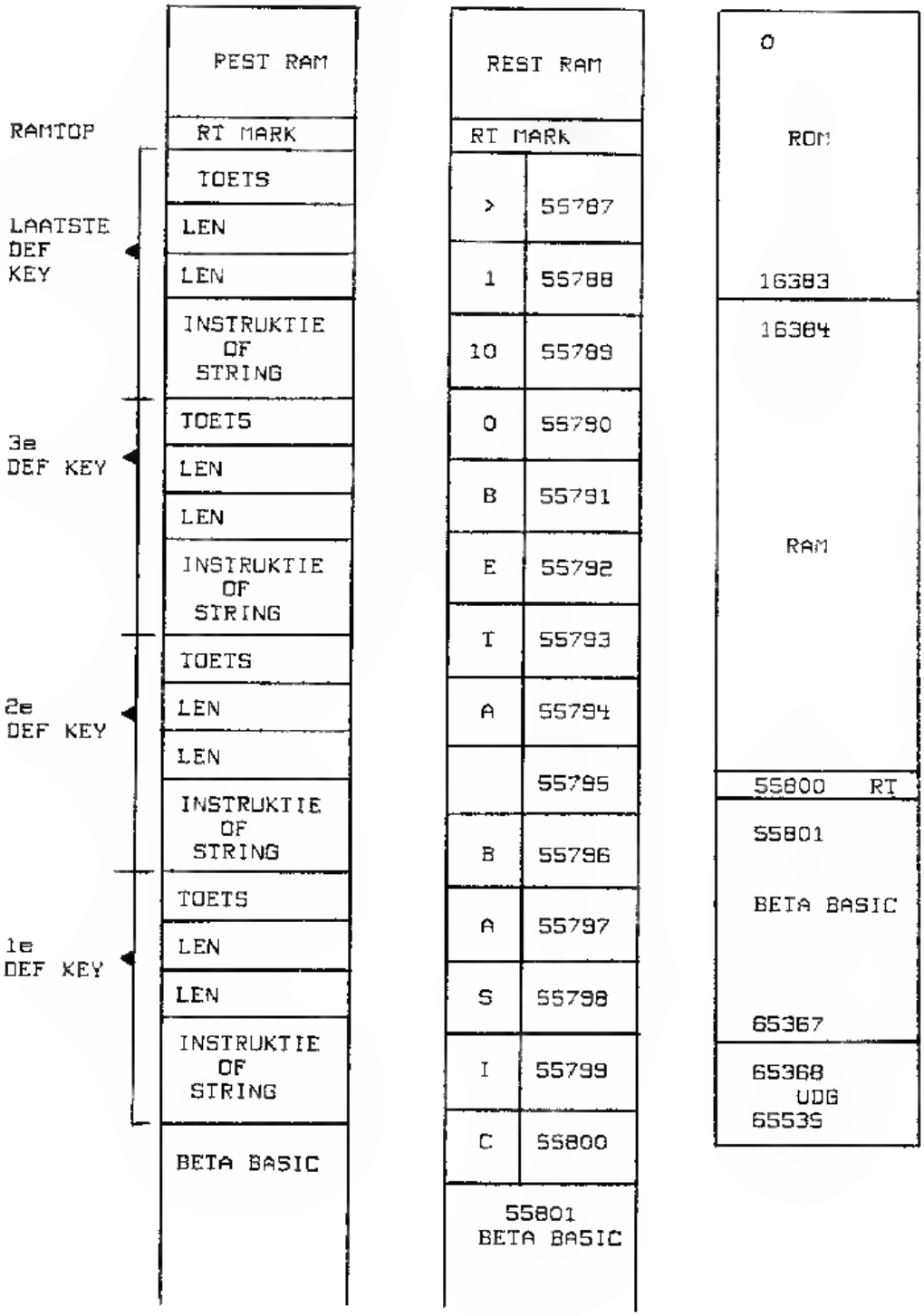
De vraagtekens die men kreeg bij PRINT CHR$ 10 en 11 zijn verdwenen.

CHR$ 13 is de 'carriage return' of nieuwe regel, indien het wordt gePRINT. Bij PLOTinstrukties wordt het CHR$ 13 een vraagteken!

VOORBEELDPROGRAMMA:

```
  5 REM CURSOR CONTROLE CODES
 10 DO
 20 GET A$
 30 IF A$=CHR$ 13 THEN PRINT "
";
 40 IF A$=CHR$ 9 OR A$=CHR$ 11
OR A$=CHR$ 10 THEN PRINT " ";CHR
$ 8;CHR$ 8;CHR$ 9;
 60 PRINT A$; FLASH 1;">"; FLAS
H 0;" ";CHR$ 8;CHR$ 8;
 70 LOOP UNTIL A$=" STOP "
 80 STOP
 90 LET A$="CURSCONTRCODE "
100 SAVE *"M";1;A$
110 VERIFY *"M";1;A$
```

GEHEUGENINDELING bij DEF KEY.



DE KARAKTERSET VAN BETA BASIC (t/m 1.8).

DEC. CODE	KEYWORDS	TOETSEN	BETA BASIC VERSIE	
128	KEYWORDS	GR 8	1.0	
129	DEF PROC	GR 1	1.0	
130	PROC	GR 2	1.0	
131	END PROC	GR 3	1.0	
132	RENUM	GR 4	1.0	
133	----			niet gebruikt
134	AUTO	GR 6	1.0	
135	DELETE	GR 7	1.0	
136	----			niet gebruikt
137	JOIN	GR SS 6	1.8	
138	EDIT	0 en GR SS 5	1.8	
139	KEYIN	GR SS 4	1.8	
140	----			niet gebruikt
141	----			niet gebruikt
142	DEF KEY	GR SS 1	1.8	
143	----			niet gebruikt
144	ALTER	GR A	1.0	
145	B			niet gebruikt
146	CLOCK	GR C	1.0	
147	DD	GR D	1.0	
148	ELSE	GR E	1.0	
149	FILL	GR F	1.0	
150	GET	GR G	1.0	
151	H			niet gebruikt
152	EXIT IF	GR I	1.0	
153	WHILE	GR J	1.0	
154	UNTIL	GR K	1.0	
155	LOOP	GR L	1.0	
156	SORT	GR M	1.0	
157	ON ERROR	GR N	1.0	
158	ON	GR O	1.0	
159	DPOKE	GR P	1.0	
160	POP	GR Q	1.0	
161	ROLL	GR R	1.0	
162	SCROLL	GR S	1.0	
163	TRACE	GR T	1.0	
164	USING	GR U	1.0	

N.B. GR= GRAPHICS
SS= SYMBOL SHIFT

Kleurcontrole codes

Voor het duidelijk en mooier maken van listings of het kleuren van strings zijn er kleurcontrole toetscombinaties met SYMBOL SHIFT en CAPS SHIFT - dit is de extended mode - voorzien. Hiermee kunnen we de attribuutgegevens; PAPER, INK , BRIGHT en FLASH en ook INVERSE en TRUE VIDEO bepalen voor een string of deel van de listing of van een keyword in de listing, zonder dat de ingebrachte toetsen in de listing (of PRINT instruktie) zichtbaar zijn.

```
10  LET A$="12345"
```

Deze string A$ heeft bv. PAPER 1 en INK 6, BRIGHT 1), terwijl de permenente attributen INK toch op 7 en PAPER 0 en BRIGHT op 0 staan. Het zijn tijdelijke kleurattributen die in a$ werden opgenomen.
Het is dan wel nuttig steeds de LEN van A$ te controleren, want die wordt groter, terwijl zulks niet te zien is aan de lengte van de string. a$="12345" hoeft niet LEN 5 te hebben : brengen we na het deel van regelnummer 10 LET A$=" de kleuroptie in. Onze mogelijkheden zijn :

```
10  LET a$="
```

KLEUR - CONTROLE TOETSCOMBINATIES .

MODE	IN TE DRUKKEN TOETSEN (LEN +1)	(LEN +1)	RESULTAAT	
E	CAPS + SYMBOL,	9	voor	BRIGHT 1
E	CAPS + SYMBOL,	8	voor	BRIGHT 0
E	CAPS + SYMBOL,	CAPS + 9	voor	FLASH 1
E	CAPS + SYMBOL,	CAPS + 8	voor	FLASH 0
E	CAPS + SYMBOL,	N	voor	PAPER 0 <= n <= 7
E	CAPS + SYMBOL,	CAPS + N	voor	INK 0 <= n <= 8.
		CAPS + 3	voor	INV.VIDEO
		CAPS + 4	voor	TRUE VIDEO

Let er op dat wanneer u in de listing in A$ met het eerste en tweede (niet zichtbare karakter) BRIGHT aan zet , de totale listing na deze karakters in BRIGHT 1 komt, zoleng niet de (onzichtbare CAPS+SYMBOL, 8) worden gelezen, welke BRIGHT weer tot zijn eerste stand terugbrengen. Dit principe geldt ook voor PAPER, FLASH en INVERSE: Wat aan kleurcontrolecodes wordt 'geopend' dient netjes 'afgesloten' anders wordt het toch een warboel.
Daarom is het best dat u de in te kleuren string of keywords of het in te kleuren deel van de listing onmiddellijk afbakent: zet je vooraan BRIGHT op, breng dan achteraan BRIGHT 0 in.

Maak je A$="(PAPER 4)12345 met a$="CAPS+SYMBOL, 4)12345
dan is A$ (IO) het best

```
A$="(CAPS+SYMB, 4)12345(CAPS+SYMB, N )"
PRINT LEN A$ :    LEN A$= 9!
```

ALGEMEEN GELDT: SLUIT AF TOT OORSPRONKELIJKE STAND.

Overzicht ven de foutboodschappen.

Hierne volgen de foutboodschappen van Spectrum- en Beta-Basic en deze ven de Interface 1 en de Discovery. De foutwaarde (in de eerste kolom) gebruikt men om de variabele ERROR zijn waarde toe te kennen. Dit geat niet op voor de boodschappen 0 en 9, met name 0 Ok. en 9 STOP-statement.

DE SPECTRUM FOUTBOODSCHAPPEN.			INTERFACE 1 FOUTBOODSCHAPPEN		
0	0	OK	43	b	Program finished
1	1	NEXT without FOR	44	c	Nonsense in BASIC
2	2	Verieble not found	45	d	Invalid stream number
3	3	Subscript wrong	46	e	Invelid device expression
4	4	Out of memory	47	f	Invelid name
5	5	Out of screen	48	g	Invelid drive number
6	6	Number too big	49	h	Invalid station number
7	7	RETURN without GOSUB	50	i	Missing name
8	8	End of file	51	j	Missing station number
9	9	STOP statement	52	k	Missing drive number
10	A	Invalid argument	53	l	Missing baud rate
11	B	Integer out of renge	54	m	Header mismatch error
12	C	Nonsense in BASIC	55	n	Stream already open
13	D	BREAK - CONT repeats	56	o	Writing to a "read" file
14	E	Out of DATA	57	p	Reading a "write" file
15	F	Invelid file neme	58	q	Drive "write" protected
16	G	No room for line	59	r	Microdrive full
17	H	STOP in INPUT	60	s	Microdrive not present
18	I	FOR without NEXT	61	t	File not found
19	J	Invelid I/O device	62	u	Hook error code
20	K	Invalid colour	63	v	CODE error
21	L	BREAK into progrem	64	w	MERGE error
22	M	RAMTOP no good	65	x	Verification has failed
23	N	Statement lost	66	y	Wrong file type
24	O	Invelid stream			
25	P	FN without DEF			
26	Q	Paremeter error			
27	R	Tape loading error			

De boodschappen m en u werden in het interface 1 handboekje niet neder omschreven.
Foutboodschappen c Nonsense in BASIC is in betekenis gelijk aan foutboodschap C ven Spectrum Basic, meer wordt alleen aangeroepen door interfece 1 verrichtingen.

BETA-BASIC FOUTBOODSCHAPPEN.

28	S	Missing LOOP	42	a	Invalid device name
29	T	LOOP without DO	43	b	Stream already open
30	U	No such line	44	c	Invalid drive number
31	V	No POP data			
32	W	Missing DEF PROC	46	e	Write protected
33	X	No END PROC	47	f	Disc full
34	Y	Too hard	48	g	Disc I/O error
			49	h	File not found
RENUM-boodschap :			50	i	Hook code error
'FAILED at lijnnummer:statement			51	j	File size error
			53	l	Verification failed
			54	m	Wrong file type
				*1	RAM corrupt
				?	no message defined

UITLEG BETA BASIC FOUTBOODSCHAPPEN.

28 S Missing LOOP

Een EXIT IF of een conditionele DO (eentje met UNTIL of WHILE) zocht een einde van de verwachte DO-LOOP, maar vond dit niet.

29 T LOOP without DO

Er werd een LOOP-instruktie gelezen, aan geen DO-statement aan vooraf ging.

30 U No such line

Bij gebruik van DELETE <lijn> TO <lijn>, terwijl een aangegeven 'lijn' niet (niet meer) in het programma voorkomt.

31 V No POP data

Met POP ward getracht een adres uit de DO-LOOP/GOSUB/ of PROC-stack te werpen, terwijl gaen adres op de stack stond (d.i. Er was geen PROC, GOSUB of DO-LOOP in werking.)

32 W Missing DEF PROC

Er werd een END PROC teveel gelezen of bij een meervoudige uitgang werd een END PROC ongewenst geldig gelezen. Een PROC naam werd gelezen, maar deze blijkt niet gedefinieerd met DEF PROC. Foute naam ingetikt!

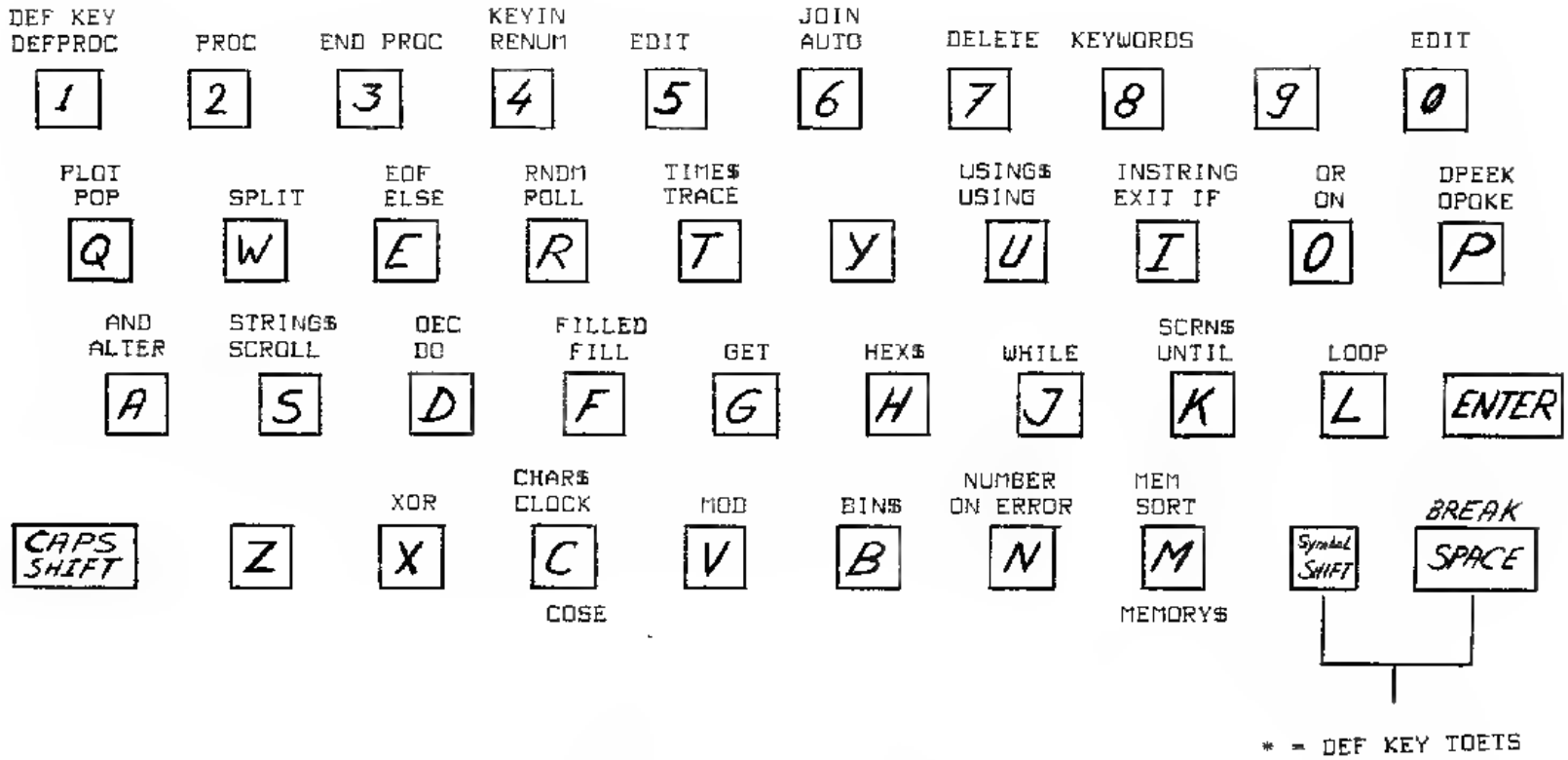
33 X No END PROC

Bij het springen over een procedure (niet tijdens het uitvoeren ervan) werd gaen (geldige) END PROC gevonden.

34 Y Too hard

Bij de programmahernummering is een lijnnummer aangege- van in de vorm van een ta moeilijke expressie (BB 1.8)

RENUM-boodschap: FAILED at lijnnummer: statament (zie RENUM 1.9)

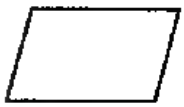


TOETSENBORD LAYOUT
VOOR BETA BASIC KEYWORDS.

GEBRUIKTE SYMBOLEN

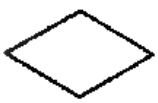
= begin/einde van programma,subroutine, procedure.

= bewerking (PRINT,FOR-NEXT,DO-LOOP)



= INPUT/OUTPUT

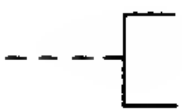
= aanroep van PROCEDURE,Subroutine,DO-LOOP



= beslissing

= bladzijde connector

* pijltjes worden alleen gebruikt als tegen de natuurlijke stroomrichting ingegaan wordt.(natuurlijk is van boven naar beneden en van links naar rechts).



= Teken om verklarende tekst aan te geven.

P R O G R A M M E E R T I P S

De allereerste fase van het programmeren is het maken van een analyse van het probleem; wat moet het programma doen, wat gaat erin en wat moet er uit komen. Complexe problemen dienen onderverdeeld te worden in deelproblemen.

De tweede fase bestaat uit het in een systeem-flowchart brengen van het gehele gebeuren. Een systeem-flowchart beschrijft globaal het totale proces. Het verloop van de gegevens moet in grote lijnen vaststaan. Indien het om eenvoudige programma's gaat kan deze fase achterwege blijven. Bij complexe programma's is het echter een vereiste.

De derde fase is het maken van programma-flowcharts aan de hand van de systeem-flowchart. Een programma-flowchert dient om een probleem in fasen te analyseren en vast te leggen en om de bewerkingen van de computer, algoritmisch, in detail weer te geven. Het maken van progromma-flowcharts zal u in het begin misschien vreemd aan doen. Het is echter een zeer goed hulpmiddel en een zeer goede gewoonte. Het zal u bovendien veel tijd besparen, omdat u reeds op papier heeft " uitgedokterd " hoe een en ander zal verlopen.

De vierde fase omvat het coderen (in BASIC vertalen) van de programme-flowchart(s) en het intikken. In veel gevallen zullen meerdere oplossingen toegepast kunnen worden. De funktie TIME$ ( ) komt hier goed van pas om verschillende oplossingen in snelheid te vergelijken.

Tracht het gebruik van GO TO zoveel mogelijk te vermijden. Beta-Basic biedt u een uitgebreid DO LOOP lussen-systeem waarmee de meeste GO TO's overbodig gemaakt kunnen worden. GOSUB's kunnen bijna altijd vervangen worden door PROCedure's te benoemen, welke bovendien sneller zijn. PROCedure's kunnen overal in het programma geplaatst worden; het is echter beter om de PROCedure's die het meest gebruikt worden (of die het snelst doorlopen moeten worden) in het begin te zetten.

Het gebruik van REM statements is tijdens het intikken aan te raden. Indien het programme naar wens verloopt, kunnen deze eruit gehaald worden om bijvoorbeeld snelheid en geheugen te winnen.

VAL " waarde " bespaart, zoals reeds bekend, veel geheugen (some tot 2 K !) maar doet het programma aanmerkelijk langzamer lopen.

Constanten (dat zijn nummerieke of string variabelen waarvan de waarde of inhoud tijdens het gehele programma ongewijzigd blijven) kunnen na te zijn ingelezen geDELETEd worden. Eventueel kunnen deze ook boven ramtop weggePOKEd worden, zodat ze bij een RUN of CLEAR behouden blijven.

ON ERROR dient tot het minimum beperkt te blijven; het programma zou in feite niet onderbroken mogen worden door foutboodscheppen veroorzaakt door "onvoorziene" omstandigheden. De onderbrekingen veroorzaakt door SAVE en LOAD (tape loading error, microdrive full ed.) zijn niet op te vangen, tenzij door gebruik van ON ERROR.

Het volgende voorbeeld toont aan hoe de lengte van een programme-naam gecontroleerd kan worden op fouten.

```
10 DO
20 INPUT " NAAM "; LINE A$
30 EXIT IF LEN A$ <= 10
40 PRINT #0; BRIGHT 1; FLASH 1; AT 1,0;" TE LANG ! ":
   BEEP .1,.1 : PAUSE 40
50 LOOP
60 SAVE A$
```

Om een verkeerde ingave van een keuze te onderscheppen kan het volgende dienen:

```
10 DO
20 PRINT #0; BRIGHT 1; AT 1,0; " ENTER UW KEUZE "
30 GET KEUZE
40 EXIT IF KEUZE <= 4
50 PRINT #0; BRIGHT 1; FLASH 1;  AT  1,0;  " VERKEERDE
   KEUZE ! ": BEEP .1,.1 : PAUSE 40
60 LOOP
70 GO TO ON KEUZE; 100 ,200 ,300 ,400
100 REM KEUZE 1
200 REM KEUZE 2
300 REM KEUZE 3
400 REM KEUZE 4
500 STOP
```

Gebruikers-informatie op het scherm (wat moet ik intikken om te stoppen) is gebruikers-vriendelijker dan wanneer er geraden moet worden naar de manier om het programma te onderbreken. Met informatie op het scherm "vertelt" de computer u wat u mag doen en als u dan toch een verkeerde toets heeft ingedrukt dan zal de computer u dat melden. Gebruik om te stoppen zoveel mogelijk dezelfde toets (de "s" bijvoorbeeld).

Het gebruik van IF ... THEN .... (zonder ELSE !) kan vermeden worden indien de voorwaarde bestaat uit ja/nee (een of nul) met behulp van de Spectrum logische operatoren AND en OR en de functie NOT. (De Bete Basic functies AND, OR en XOR doen in feite hetzelfde op binair niveau).

Enkele voorbeelden:

```
10 DO
20 LET A= RNDM(20)
30 PRINT A;
40 REM A WORDT VERLAAGD MET 10 ALS A KLEINER OF
   GELIJK IS AAN 10
50 LET A= A - (10 AND A<= 10)
60 PRINT TAB 6;A
70 LOOP
```

Regel 50 vraagt wellicht om enige uitleg. U kunt als directe opdracht geven:

```
PRINT 1=1
```

De Spectrum zal antwoorden met: 1 (oftewel: ja). Geef nu:

```
PRINT 1=2
```

De Spectrum zal nu antwoorden met: 0 (oftewel: nee)

Gebruikmakend van deze wetenschap kunnen wij dus stellen :
LET A = A (en-10 als A kleiner dan of gelijk is een 10) Als A een waarde bereikt heeft van 10 of groter dan zal er 10 van af getrokken worden, indien niet dan blijft A zoals het was.

Een gelijkeerdig voorbeeld:

```
10 DO
20 LET X= RNDM(1)
30 PRINT ("EEN" AND X)+("NUL" AND NOT X);" ";X
40 POKE 23692,255
50 EXIT IF INKEY$= " STOP "
60 LOOP
70 STOP
```

Regel 30 zal "EEN" PRINTen als X = 1 en zal "NUL" PRINTen als X=0.

Met OR:

```
10 LET A =10
20 DO
30 PRINT A;
40 REM A WORDT OP 1 GEZET INDIEN MET KLEINER DAN 1
   DREIGT TE WORDEN
50 LET A= A OR (A<1)
60 PRINT TAB 6;A
70 LET A = A -1
80 LOOP
```

Regel 50 zet A op 1 als het kleiner dan 1 is geworden.

Met strings:

```
10 DO
20 LET A$=" BETA BASIC"
30 LET B = RNDM (1)
40 PRINT A$ AND B
50 LOOP
```

Regel 40 wordt uitgevoerd als B = 1, Als B = 0 wordt er niets gePRINT.
Dus: (als B= 1) PRINT A$ AND B (=1, dus PRINT maar)
en (als B= 0) PRINT A$ AND B (=0, dan niet PRINTen)

DELETE en MERGE zijn zeer handig te gebruiken als U de beschikkinh heeft over 1 (of meerdere) microdrives of discdrives, U dient den echter wat men noemt "modulair" te programmeren, wet onder andere inhoud det bepeelde zeken een degelijke organisatie vereisen. U kunt van ellerlei hendige routine's een "bibliotheek" eenleggen, waarven de regelnummers bijvoorbaald eltijd venef 9000 baginnen, Door middel van een geDIMensioneerde errey kunt u de nemen van de programma's op een menu zetten en een keuze meken ven wet er geMERGEd moet worden.

Het programme weerin de DATA wordt opgebouwd behoeft den niet meer gebruikt te worden.

Voorbeeld om de DATA op te bouwen:

```
10 DIM A$( 5,10)
20 LET A$(1)= "subr 1"
30 LET A$(2)= "subr 2"
40 LET A$(3)= "subr 3"
50 LET A$(4)= "subr 4"
60 LET A$(5)= "subr 5"
70 SAVE *"M";1;  "MERGEPROGR" DATA A$()
```

De MERGE routine's zelf dienen bij het SAVEn een vaste LEN van 10 te hebben anders krijgt u " file not found " omdat de LEN verschillend kan zijn.

Bijvoorbeeld:

```
10 DIM X$(10)
20 LET X$="subr 1"
30 SAVE X$
```

In uw programma gebruikt u dan:

```
  10 uw programma
7999 restant
8000 DEF PROC MERGEMENU
8005 LOAD *"M";1;"MERGEPROGR" DATA A$()
8010 PRINT "MERGE MENU"
8020 FOR F = 1 TO 5
8030 PRINT AT F*2,0;F;"-";  A$(F)
8040 NEXT F
8050 PRINT AT 12,0;" 6-STOP"
8100 PRINT AT 21,0;" MAAK KEUZE"
8110 DO
8120 GET KEUZE
8130 EXIT IF KEUZE <= 6
8140 LOOP
8150 MERGE A$(KEUZE)
8160 INPUT " het geMERGEde achteraf DELETEn ? (J/N)  ";
     LINE Z$
8170 GOSUB 9000: REM DE ROUTINE MOET AAN HET EINDE EEN
     " RETURN " HEBBEN !!
8180 IF Z$="J" OR Z$="j" THEN DELETE 9000 TO
8200 END PROC
```

Let erop dat de routines eindigen met een RETURN naar de procedure. In uw eigen programma's dient u de menu's aan te passen dat deze de PROCedure MERGEMENU aanroepen en er weer naar terugkeren en bovendien moet u de mogelijkheid in het menu voorzien dat een reeds eerder geMERGEd programma kan aangeroepen worden met GOSUB 9000. U behoeft dan niet weer opnieuw te MERGEn.

De laatste fase bestaat uit het testen van het programma of de procedure's. Indien blijkt dat een en ander niet zoals gewenst verloopt kunnen de betreffende programma-flowcharts geraadpleegd worden en hulp biedan met daarnaast de listing van het programma (op scherm of op papier).TRACE kan dan goed gebruikt worden om de programma-stroom af te drukken op het scherm of op de printer, zodat u inzicht krijgt in het verloop van het programma.

BETER PROGRAMMEREN MET BETA-BASIC
EN ZX SPECTRUM (+)

Dit boek is een uitgebreid HANDBOEK geworden voor gebruik van een der krachtigste BASICs ter wereld.
Omdat de mogelijkheden zo uitgebreid zijn, voldoet de handleiding die bij het software-pakket geleverd wordt in het geheel niet. Dit werd ook zo ondervonden door beide auteurs. Zij schreven dit handboek om de vele gebruikers van BETA-BASIC 1.0, 1.8 en 1.9 de vele mogelijkheden eens duidelijk uit te leggen aan de hand van vele tientallen programmavoorbeelden.
Naast de uitleg van de KEYWORDS en FUNCTIES zijn er ook nog een aantal tabellen opgenomen, alsmede een aantal uitgebreide programma's, die gebruik maken van BETA-BASIC.
We noemen o.a. DATABASE, I TJING, DEF-KEY, SCREENS EN VARIABELENINFO.

Een onmisbaar naslagwerk voor hen die alles uit BETA-BASIC willen halen.

TERMINAL SOFTWARE PUBLICATIES

TERMINAL

BETER PROGRAMMEREN MET BETA-BASIC
EN ZX SPECTRUM (+)

Dit boek is een uitgebreid HANDBOEK geworden voor gebruik van een der krachtigste BASICs ter wereld.
Omdat de mogelijkheden zo uitgebreid zijn, voldoet de handleiding die bij het software-pakket geleverd wordt in het geheel niet. Dit werd ook zo ondervonden door beide auteurs. Zij schreven dit handboek om de vele gebruikers van BETA-BASIC 1.0, 1.B en 1.9 de vele mogelijkheden eens duidelijk uit te leggen aan de hand van vele tientallen programmavoorbeelden.
Naast de uitleg van de KEYWORDS en FUNCTIES zijn er ook nog een aantal tabellen opgenomen, alsmede een aantal uitgebreide programma's, die gebruik maken van BETA-BASIC.
We noemen o.a. DATABASE, I TJING, DEF-KEY, SCREENS EN VARIABELENINFO.

Een onmisbaar naslagwerk voor hen die alles uit BETA-BASIC willen halen.

TERMINAL SOFTWARE PUBLICATIES

BETER PROGRAMMEREN MET BETA-BASIC EN ZX SPECTRUM Terminal

Beter programmeren met BETA-BASIC en ZX SPECTRUM (+)

S. Girard
L. Verboven

TERMINAL SOFTWARE PUBLICATIES

Beter programmeren met BETA-BASIC en ZX SPECTRUM (+)

S. Girard
L. Verboven

TERMINAL SOFTWARE PUBLICATIES